

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.

Содержание

Аннотация.....	3
Постоянный модуль	4
Блок А: ШИФРЫ ОДНОЗНАЧНОЙ ЗАМЕНЫ	5
1. Шифр простой замены АТБАШ.....	5
2. ШИФР ЦЕЗАРЯ	7
3. Квадрат Полибия.....	9
Блок В: ШИФРЫ МНОГОЗНАЧНОЙ ЗАМЕНЫ	12
4. Шифр Тритемия	12
5. Шифр Белазо	15
6. Шифр Виженера.....	18
Блок С: ШИФРЫ БЛОЧНОЙ ЗАМЕНЫ.....	21
8. Матричный шифр.....	21
9. Шифр Плейфера	24
Д: ШИФРЫ ПЕРЕСТАНОВКИ	29
10. Шифр вертикальной перестановки	29
11. Решетка Кардано.....	32
Е: ШИФРЫ ГАММИРОВАНИЯ	36
13. Одноразовый блокнот К.Шеннона.....	36
14. Гаммирование ГОСТ 28147-89.....	41
Ф: ПОТОЧНЫЕ ШИФРЫ.....	46
15. А5 /1	46
16. А5 /2.....	54
Блок G: КОМБИНАЦИОННЫЕ ШИФРЫ.....	62
17. МАГМА.....	62
БЛОК Н: АСИММЕТРИЧНЫЕ ШИФРЫ.....	68
21. RSA	68
22. Elgamal	72
Блок I: АЛГОРИТМЫ ЦИФРОВЫХ ПОДПИСЕЙ.....	86
24. RSA	86
25. El Gamal	89
Блок J: СТАНДАРТЫ ЦИФРОВЫХ ПОДПИСЕЙ.....	92
26. ГОСТ Р 34.10-94.....	92
27. ГОСТ Р 34.10-2012.....	95
28. ОБМЕН КЛЮЧАМИ ПО ДИФФИ-ХЕЛЛМАНУ	100

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 = "абвгдеёжзийклмнопрстуфхцчщъыьэюя"

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)
```

Блок А: ШИФРЫ ОДНОЗНАЧНОЙ ЗАМЕНЫ

1. Шифр простой замены АТБАШ

Атбаш — простой шифр подстановки для алфавитного письма. Правило шифрования состоит в замене i -й буквы алфавита буквой с номером $n-i+1$, где n — число букв в алфавите.

Код программы:

```
from base import alphabet, input_for_cipher_short, input_for_cipher_long,
output_from_decrypted

def atbash(input):
    return input.translate(str.maketrans(
        alphabet + alphabet.upper(), alphabet[::-1] + alphabet.upper()[::-1]))

print(f'''
ШИФР АТБАШ:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{atbash(input_for_cipher_short())}

Расшифрованный текст:
{output_from_decrypted(atbash(atbash(input_for_cipher_short())))}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{atbash(input_for_cipher_long())}

Расшифрованный текст:
{output_from_decrypted(atbash(atbash(input_for_cipher_long())))}
''')
```

Тестирование:

```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab01_1_atbash.py
ШИФР АТБАШ:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
эоътачпмпоцуцэдцрмуцэдсъшылмзъурэъфямзф

Расшифрованный текст:
время, приливьиотливынеждутчеловека.

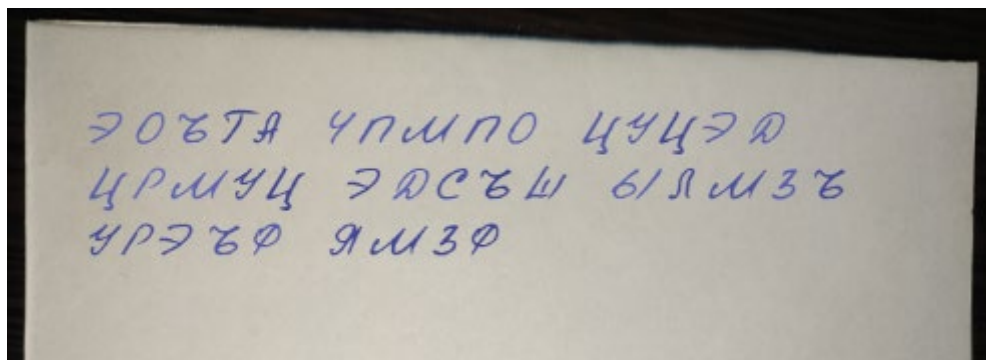
ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
эрмпоцтъонмямгцсямдназлнцтэрурэмзфвмрырнмямрзсртяуъсгфцхмъфнмчпмрпмцтяугсрпрый
рыаёцхыуафяомрзъфмрээрээцсмъосъмцуцтяьячцсаяцуцыуасъюругжцийцскротяицрссдйплюу
цфяицхмзфэмяфртмъфнмъоъфрюдэяъмюруъыэлйцуцмощйяючяиъэцрюдзсррыцспрычяърурэрф
мзфсртршсрцуючсърмзфсмямназлнцтэрурэрэоъфртъсырээрсрцнпругчрэмгрыцсцуцыэяфубзяц
рыслфяомцслмзфмъфнмсямдназлнцтэрурэрвмрнфругфрпоцтъосрнурэмзфнмямцнмцфяпрфячдэя
ъмчпмзмрмдназаяэфубзяъмэнъюанмрпамгыънамцуцыэъмнцнурэноъысхъуцзсдмзфсрчпмъну
цчурлпрмоъюуамгпоъуурьятцчпмнрбчятццыольцтцзянматцоъзцсярыцсцуцыэянцтэруячпмр
```

фруцзънмэрнурэсъчттьсрэрчоянмяъммзфэфрпцояхмъонфрхыъамъугсрнмцпоцсамрнзцмямгм
дназцнпорюуятццуцуючмзфлзъмпорюурэлэъуцзцэяъмрюетмъфнмяпоцтьосрсянмрцуцыэън
мнцтэрурэцттьсрнмругфроячтдоачыъуаътнурэянэрюрысдтпорнмояснмэртмзфнзцмямгпорю
ъудчяфячзцфцсъубюамчпммяффяфвмрплнмрътътнмрмзфрысяфрсъфрмродъкцотдцюошцэцыамнп
ояэъуцэдтнмяэцмгнмрцтрнмгчямдназлнцтэрурэнпорюуятццпмнзцмяапрнуъысцъэяшсдтву
ътъсмртфязънмэъсрърэрнпоцамцамзфнръуянцмънгчпмзцмямгнущмсдхмъфнмюъчъыщсрърпор
плнфячпмсцфмрсъюлыъммзфсррюугжцснмэлслшсяиъсячямдназлчсфрэюъчпорюурэмзф

Расшифрованный текст:

вот пример статьи на тысячу символов. это достаточно маленький текст, оптимально подходящий для карточек товаров в интернет или магазина 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 картинку. текст на тысячу символов это сколько примерно слов. статистика показывает, что тысяча включает в себя столько десяти или двести слов средней величины. но, если злоупотреблять предложениями, союзами и другими частями речи на один или два символа, то количество слов не изменно и возрастает. в копирайтерской деятельности принято считать тысячу пробелами или без. учет пробелов увеличивает объём текста примерно на сто или двести символов именно столько раз мы разделяем слова свободным пространством. считать пробелы заказчики не любят, так как это неустойчивое место. однако некоторые фирмы биржи видят справедливым ставить сто тысяч 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.

Карточка:



2. ШИФР ЦЕЗАРЯ

Шифр Цезаря, также известный как шифр сдвига, код Цезаря или сдвиг Цезаря — один из самых простых и наиболее широко известных методов шифрования.

Шифр Цезаря — это вид шифра подстановки, в котором каждый символ в открытом тексте заменяется символом, находящимся на некотором постоянном числе позиций левее или правее него в алфавите.

Код программы:

```
from base import alphabet, input_for_cipher_short, input_for_cipher_long,
output_from_decrypted
import random

key = 5

def caesar_encode(input, step):
    return input.translate(
        str.maketrans(alphabet, alphabet[step:] + alphabet[:step]))

def caesar_decode(input, step):
    return input.translate(
        str.maketrans(alphabet[step:] + alphabet[:step], alphabet))

print(f'''
ШИФР ЦЕЗАРЯ:
Ключ: {key}
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{caesar_encode(input_for_cipher_short(), key)}

Расшифрованный текст:
{output_from_decrypted(caesar_decode(caesar_encode(
    input_for_cipher_short(), key), key))}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{caesar_encode(input_for_cipher_long(), key)}

Расшифрованный текст:
{output_from_decrypted(caesar_decode(caesar_encode(
    input_for_cipher_long(), key), key))}
''')
```

Тестирование:

```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab01_2_caesar.py

ШИФР ЦЕЗАРЯ:
Ключ: 5
КОРОТКИЙ ТЕКСТ:
```

Зашифрованный текст:

жхйсдмфчфхнрнжанучрнжатилишчъйружйпечъп

Расшифрованный текст:

время, прилив и отлив вынеждут человека.

ДЛИННЫЙ ТЕКСТ:

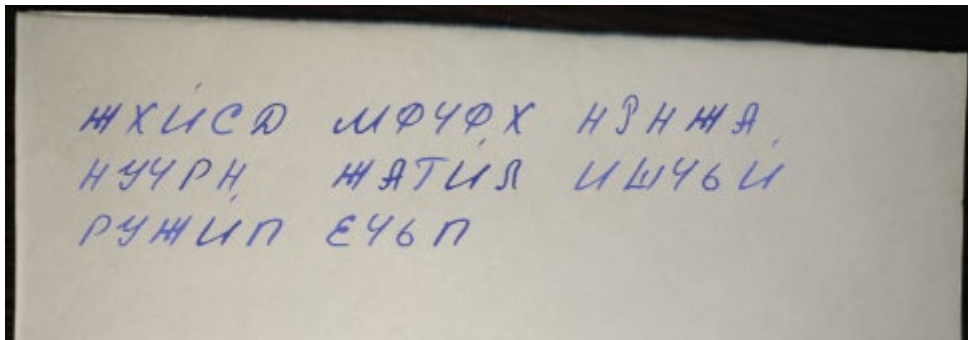
Зашифрованный текст:

жучфхнсийхччечбнтечацдъщцнсжуружчъпвчуиуцчечуътусерйтбпнойпцчмфчуфчнсербтуфуиъ
уидюноирдпехчуъйпчужехужнтчйхтйчнрнсеземнтеърнирдтйёурбэнънтшухсеынуттаъфшёр
нпечночьпжчепусчийпцйхийипуёажейчёрыйишънрнчхкъеемейжнуёаьтууинтфуимезуружуп
чъптусултунёймтйзучъптечацдъщцнсжуружхйпусйтиужетунцфурбмужечбуинтнрнижепргъен
уитшпехчнтшъпчйпцчтечацдъщцнсжуружвчуцпурбпуфхнсийхтуцружчъпччечнцнпепфупемаже
йчмфчъчучацдъежпргъейчжйёдцчуфдчбийцдчнрнижйцнцружцхйитйожйрнънтачъптумфчйцр
нмрушфучхийёрдчбфхйирузеснмфцугмесннихшзнсньецдснхийнтеуинтнрнижецнсжуремфччу
пурньйцжужуцружтйнмситтужумхецчейчъпжпуфнхеочйхцпуойидчйрбтуцнфхнтдчуънчечбч
ацдънцфхуёйресннрнёймчъпшьйчфхуёйружшжйрнънжейчуёяйсчйпцчефхнсийхтутецчунрнижйц
чнцнсжуружнситтучурбпухемсахемийрдйсружецжужёуитасфхуцчхетцчжусчъпцънчечбфхуё
йрамепемьппнтйргёдчмфччеппепвчуфшцчуйсийцчучъпуйтепутйпучухайшнхсанёнхлнжидцф
хежйирнжасцчежнбцчунсуцбмечацдъщцнсжуружцфхуёйреснмфчънчедфуцрйитнйжелтасвр
йсйтчуспёйцжйттузужуцфхндчндчъпцузрецнчйцбмфчънчечбцрнчтаочйпцчёймйинтузуфху
фшцпемфчтнпчутйёшийчъптуёурбэнтцжштшлтеыйтемечацдъшмтепужёймфхуёйружчъп

Расшифрованный текст:

вот пример статьи на тысячу символов. это достаточно маленький текст, оптимально подходящий для карточек товаров в интернет-магазине 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 картинку. текст на тысячу символов это сколько примерно слов. статистика показывает, что тысяча включает в себя столько десяти или двести слов средней величины. но, если злоупотреблять предложениями, союзами и другими частями речи на один или два символа, то количество слов не изменно и возрастает. в копирайтерской деятельности принято считать тысячу пробелами или без пробелов. увеличивать объём текста примерно на сто или двести символов именов столько раз мы разделяем слова свободным пространством. считать пробелы заказчики не любят, так как это неустойчивое место. однако некоторые фирмы 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.

Карточка:



3. Квадрат Полибия

Квадрат Полибия – метод шифрования текстовых данных с помощью замены символов, впервые предложен греческим историком и полководцем Полибием.

К каждому языку отдельно составляется таблица шифрования с одинаковым (не обязательно) количеством пронумерованных строк и столбцов, параметры которой зависят от его мощности (количества букв в алфавите). Берутся два целых числа, произведение которых ближе всего к количеству букв в языке — получаем нужное число строк и столбцов. Затем вписываем в таблицу все буквы алфавита подряд — по одной на каждую клетку. При нехватке клеток можно вписать в одну две буквы (редко употребляющиеся или схожие по употреблению).

Код программы:

```
from base import alphabet, input_for_cipher_short, input_for_cipher_long,
output_from_decrypted

hard_dictionary = {"a": "11", "б": "12", "в": "13",
                  "г": "14", "д": "15", "е": "16", "ё": "21",
                  "ж": "22", "з": "23", "и": "24", "й": "25",
                  "к": "26", "л": "31", "м": "32", "н": "33",
                  "о": "34", "п": "35", "р": "36", "с": "41",
                  "т": "42", "у": "43", "ф": "44", "х": "45",
                  "ц": "46", "ч": "51", "ш": "52", "щ": "53",
                  "ъ": "54", "ы": "55", "ь": "56", "э": "61",
                  "ю": "62", "я": "63"}

def square_encode(input):
    new_txt = ""
    for x in input:
        if x in hard_dictionary:
            new_txt += hard_dictionary.get(x)
        else:
            new_txt += (x + x)
    return new_txt

def square_decode(input):
    new_txt = ""
    list_fraze = []
    step = 2
    for i in range(0, len(input), 2):
        list_fraze.append(input[i:step])
        step += 2
    key_hard_dictionary_list = list(hard_dictionary.keys())
    val_hard_dictionary_list = list(hard_dictionary.values())

    for x in list_fraze:
        if x in val_hard_dictionary_list:
            i = val_hard_dictionary_list.index(x)
            new_txt += key_hard_dictionary_list[i]
        else:
```

```

        new_txt += x[0:1]
    return new_txt

print(f'''
КВАДРАТ ПОЛИБИЯ:
Ключ: {hard_dictionary}
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{square_encode(input_for_cipher_short())}

Расшифрованный текст:
{output_from_decrypted(square_decode(square_encode(
    input_for_cipher_short())))}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{square_encode(input_for_cipher_long())}

Расшифрованный текст:
{output_from_decrypted(square_decode(square_encode(
    input_for_cipher_long())))}
''')

```

Тестирование:

```

/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab01_3_square.py

КВАДРАТ ПОЛИБИЯ:
Ключ: {'а': '11', 'б': '12', 'в': '13', 'г': '14', 'д': '15', 'е': '16', 'ё': '21', 'ж': '22', 'з': '23', 'и': '24', 'й': '25', 'к': '26', 'л': '31', 'м': '32', 'н': '33', 'о': '34', 'п': '35', 'р': '36', 'с': '41', 'т': '42', 'у': '43', 'ф': '44', 'х': '45', 'ц': '46', 'ч': '51', 'ш': '52', 'щ': '53', 'ъ': '54', 'ы': '55', 'ь': '56', 'э': '61', 'ю': '62', 'я': '63'}
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
133616326323354235362431241355243442312413553316221543425116313413162611425126

Расшифрованный текст:
время, прилив выиотлив вынеждут человека.

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
133442353624321636414211425624331142554163514341243213343134134251266142341534
414211423451333432113116335626242542162641422335423435422432113156333435341545
341563532425153163261136423451162642341311363413132433421636331642243124321114
112324331145243124153163331612343156522445243344343632114624343333554535431231
242611462425425126134211263432421626414216361615263412551311164212343116161513
434524312442362145111223114616132434125551333434152433353415231114343134133426
425126333432342233342412162333161434425126331142554163514341243213343134133616
263432163315341311333424413534315623341311425634152433243124151311263162511124
341533432611364224334342512642162641423311425541635143412432133431341361423441
263431562634353624321636333441313413425126414211422441422426113534261123551311

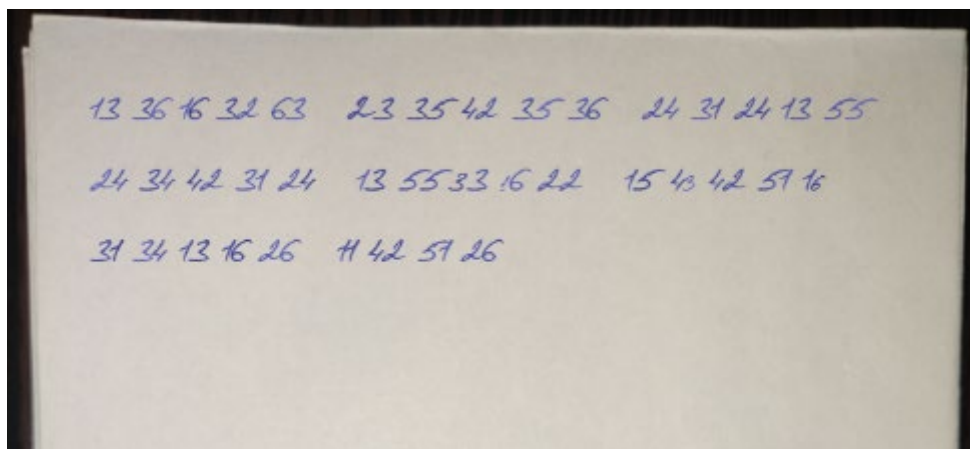
```

164223354251423442554163511113263162511116421341161263414234356342561516416342
243124151316414224413134134136161533162513163124512433554251263334233542164131
242331344335344236161231634256353616153134141132242335424134622311322424153643
142432245111414263322436165124331134152433243124151311412432133431112335424234
26343124511641421334413134133316242332163333413342336114142111642425126132634
352436112542163641263425151663421631563334414224353624336342344151244211425642
554163512441353634121631113224243124121623425126435116423536341216313413431316
312451241311164234125416324216264142113536243216363334331141423424312415131641
422441243213343134132432163333344142343156263436112332553611231516316316324131
341311411334123415335532353634414236113341421334324251264151244211425635363412
163155231126112351242624331631621263422335424211262611266142343543414234163216
414234425126341533112634331626344234365516442436325524122436222413241563424135
361113161531241355324142111324425641423424323441425623114255416351434124321334
313413413536341216311132242335424151244211633534413116153324161311223355326131
163216334234322611511641421316333334143413344135362463422463425126413414311141
244216415623354251244211425641312442335525421626414212162316152433341434353634
354341261123354233242642343316124315164242512633341234315652243341421343334322
33114616331123114255416351432333112634131216233536341216313413425126

Расшифрованный текст:

вот пример статьи на тысячу символов. это достаточно маленький текст, оптимально подходящий для карточек товаров в интернет или магазина 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 картинку. 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 предложениями, союзами и другими частями речи на один или два символа, то количество слов не изменно и возрастает. в копирайтерской деятельности принято считать тысячу пробелами или без учета пробелов увеличивать объем текста примерно на сто или двести символов именно столько раз мы разделяем слова свободным пространством. считать пробелы заказчики не любят, так как это неустойчивое место. одна из некоторых фирм биржи видит 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.

Карточка:



Блок В: ШИФРЫ МНОГОЗНАЧНОЙ ЗАМЕНЫ

4. Шифр Тритемия

Шифр Тритемия предполагал использование алфавитной таблицы. Он использовал эту таблицу для многоалфавитного зашифрования самым простым из возможных способов: первая буква текста шифруется первым алфавитом, вторая буква — вторым и т. д. В этой таблице не было отдельного алфавита открытого текста, для этой цели служил алфавит первой строки. Таким образом, открытый текст, начинающийся со слов HUNC CAVETO VIRUM ..., приобретал вид HXPF GFBMCZ FUEIB

Преимущество этого метода шифрования по сравнению с методом Альберти состоит в том, что с каждой буквой задействуется новый алфавит. Альберти менял алфавиты лишь после трех или четырех слов. Поэтому его шифртекст состоял из отрезков, каждый из которых обладал закономерностями открытого текста, которые помогали вскрыть криптограмму. Побуквенное зашифрование не дает такого преимущества. Шифр Тритемия является также первым нетривиальным примером периодического шифра. Так называется многоалфавитный шифр, правило зашифрования которого состоит в использовании периодически повторяющейся последовательности простых замен.

Код программы:

```
from base import alphabet, input_for_cipher_short, input_for_cipher_long,
output_from_decrypted

def trithemius_decode(input):
    decode: str = ""
    k = 0
    for position, symbol in enumerate(input):
        index = (alphabet.find(symbol) + k) % len(alphabet)
        decode += alphabet[index]
        k -= 1
    return decode

def trithemius_encode(input):
    encode = ""
    k = 0
    for position, symbol in enumerate(input):
        index = (alphabet.find(symbol) + k) % len(alphabet)
        encode += alphabet[index]
        k += 1
    return encode

print(f'''
Шифр Тритемия:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{trithemius_encode(input_for_cipher_short())}

Расшифрованный текст:
{output_from_decrypted(trithemius_decode(trithemius_encode(
```

```
input_for_cipher_short()))))}
```

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

```
{trithemius_encode(input_for_cipher_long())}
```

Расшифрованный текст:

```
{output_from_decrypted(trithemius_decode(trithemius_encode(
    input_for_cipher_long())))}
'''
```

Тестирование:

```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-
1/lab02_4_trithemius.py
```

Шифр Тритемия:

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

всжпгмхщцщтцфоицхгэхпгыюьмтбимбелвхып

Расшифрованный текст:

время, прилив и отливы не ждут человека.

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

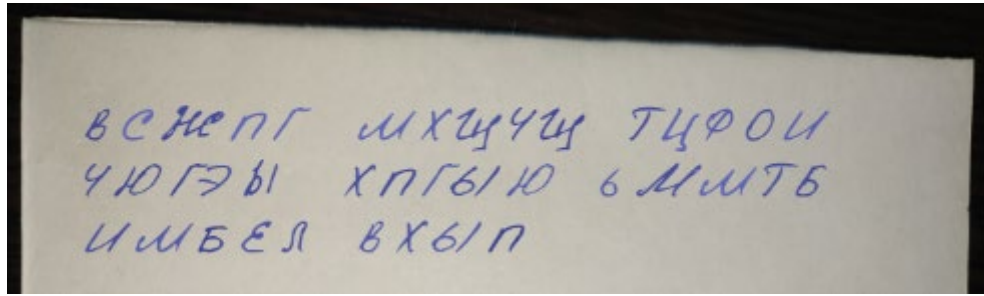
впффтфнтлшъькюицьпгмдтлизаеижжярцкюфсзучщзышвшъьоыхяюэяиьгкмгтлпмотйогпбухчна
ърмзшъхютяхжйжряёолаярпдемтшлщцэфшцыпусъьвхладвжыкгаомюымъофъчъчлгцээюмзгзцв
агшрдёпхйвувнтсшлтъьпъссшюсмфущзчёдъяяюрузлйфуйъёзпиапнхъпкзъябвшюджджэвыялйнвпм
хыпухфчршъхоучюцхвжмбешлхмыфсриндспузчмушчръсэсрябъёеегфбиэъпъндйплпнийизухигмц
эукёзуезяеллсёовиртовхяцеюътчныщэсндбеядвугзйлейгпнпуотжешъиуэяцщпаэуършчэл
вкофрнтъувывезсужбкряпафсрдёгехйфэямыпжкиедзхчошучлььссьфъучязмяеулсёйлёотёуомм
схышшэъсоъогнвдщцвъшьщждмрройгрнохшьмушхеобгряеънаёшияекжкиедгхнтицфйтыяэъя
эыкшжишжкёйюкгззнжрсузхпшйъюмтбвъъфюгеязшгмамоинйежвцйстхчыэфъючэпдбелюичкхмцн
ьхзсртсейсжфстцфнтцвъъёкшзёзжъяжкясбемкъмжёлъчкерщауъвдтгеюгыжиийорнимкжёйчшр
ътчныщэсщючбвдзйжецёкнюжмътгликтжнцьчыпобтаувшгсдзиймонюмсудсрчэсщатэляйюаятчя
пэцвшбсджелддлягвцмхщъифхлчкбюаедёгъгъйлинузмгнбмссйрхъчъёрбцялъуъеъящбэщнмоя
вёёомжбущймфйяавяъявёзтяшлхмыфсриндспузщцшлршнышшбеелюиччиимзвмиёдгктбueфоч
ръаэъысйшёзшьёжиюкямрпсквцнещяуъщошнашгцдпъеиоблъншойзтоэмцйршъйотррцьюуавдгей
игтшвкокжйтппзешйлъыбхщыоэымйыбёзшкёмбиьугмаивхяцеюърчкцыппфшбгвхъвъёлсё

Расшифрованный текст:

вот пример статьи на тысячу символов. это достаточно маленький текст, оптимально подходящий для карточек товаров в интернет-магазине или для небольших информационных публикаций. в таком тексте редко бывает более двух или трёх абзацев и обычно один подзаголовок. но можно и без него. тысячу символов рекомендовано использовать одним или двумя ключами и одну картинку. 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 предложениями, союзами и другими частями речи, на один или два символа, то количество слов не изменно и возрастает. в копирайтерской деятельности принято считать тысячу пробелами или без учёта пробелов увеличивать объём текста примерно на сто или двести символов. именно столько раз мы разделяем слова свободным пространством. считать пробелы заказчики не любят, так как это неустойчивое место. одна из некоторых фирм биржи видит справедливым ставить стоимость за тысячу

символов пробелами, считая последние важным элементом качественного восприятия. согласитесь, читать слитный текст без единого пропуска, никто не будет. но большинству нужна ценная азбука знаков без пробелов.

Карточка:



5. Шифр Белазо

В 1553 Джованни Баттиста Белазо предложил использовать для многоалфавитного шифра буквенный, легко запо-минаемый ключ, который он назвал паролем. Паролем могло служить слово или фраза. Пароль периодически записывался над открытым текстом. Буква пароля, расположенная над буквой текста, указывала на алфавит таблицы, который исполь-зовался для зашифрования этой буквы. Например, это мог быть алфавит из таблицы Тритемия, первой буквой которого являлась буква пароля. Однако Белазо, как и Тритемий, использовал в качестве алфавитов шифра обычные алфавиты.

Код программы:

```
from base import alphabet, input_for_cipher_short, input_for_cipher_long,
output_from_decrypted

key = 'ключ'

def bellaso_decode(input, key):
    decrypted = ''
    offset = 0
    for ix in range(len(input)):
        if input[ix] not in alphabet:
            output = input[ix]
            offset += -1
        elif (alphabet.find(input[ix])) > (len(alphabet) -
(alphabet.find(key[((ix + offset) % len(key))])) - 1):
            output = alphabet[(alphabet.find(
                input[ix]) - (alphabet.find(key[((ix + offset) % len(key))])))
% 33]
        else:
            output = alphabet[alphabet.find(
                input[ix]) - (alphabet.find(key[((ix + offset) % len(key))]))]
            decrypted += output
    return decrypted

def bellaso_encode(input, key):
    encoded = ''
    offset = 0
    for ix in range(len(input)):
        if input[ix] not in alphabet:
            output = input[ix]
            offset += -1
        elif (alphabet.find(input[ix])) > (len(alphabet) -
(alphabet.find(key[((ix + offset) % len(key))])) - 1):
            output = alphabet[(alphabet.find(
                input[ix]) + (alphabet.find(key[((ix + offset) % len(key))])))
% 33]
        else:
            output = alphabet[alphabet.find(
                input[ix]) + (alphabet.find(key[((ix + offset) % len(key))]))]
            encoded += output
    return encoded
```

```

print(f'''
Шифр Белазо:
Ключ: {key}
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{bellaso_encode(input_for_cipher_short(), key)}

Расшифрованный текст:
{output_from_decrypted(bellaso_decode(bellaso_encode(
    input_for_cipher_short(), key), key))}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{bellaso_encode(input_for_cipher_long(), key)}

Расшифрованный текст:
{output_from_decrypted(bellaso_decode(bellaso_encode(
    input_for_cipher_long(), key), key))}
''')

```

Тестирование:

```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab02_5_bellaso.py
```

```

Шифр Белазо:
Ключ: ключ
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
мъгдйунйъжгунщащюйамжльспсйврйёмричэги

```

```

Расшифрованный текст:
время, прилив и отливы не ждут человека.

```

```

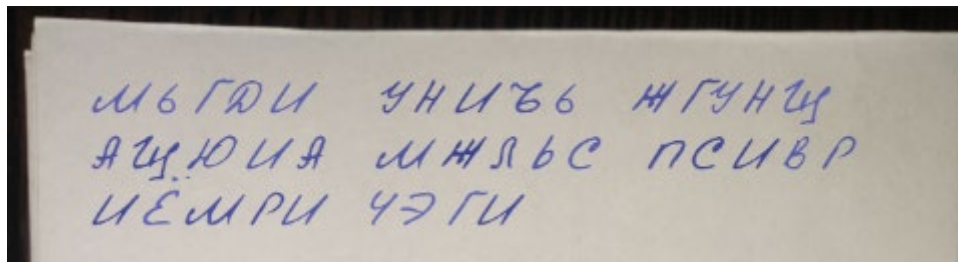
ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
мържыфкьыэрчэзжекюшийгсиушаёцъайвцыйщпмиэлрёвщмдкчгежцжбэриизунйщырачлийушънёоб
мыйежбочэвкрьёврийщнющнаашюгзшрацфкчнлёашлуацфвгйщгшщчъпубжеяъодквжёшщщмъяяг
уцунухрохнрчхъкйпцпйпъгыхъятмлгйлъйппакафйаэьдмкмёчбраашмшошъмыушнёоуюъщмшщц
рохщмдщтлёмгяшрбёэгиекюшийгсиушаёцъазпцмдпщвёмллёуэнёцзёёмлрущпжеучжымлитигтюа
щплкхлойущсйвцрьхэрекушийгсиушаёцъафэъпвщчъвшьюачроещэйёмюхвъююуэрахлнёхлётмл
гйтыроэъртъкхчмцйхвлгймэгшйэрёъркуорпцэфйаонгизфпгшнпзппльфнггуггжеёюхвшъёжэрпг
ууйёюымйръягйюъжырвгшоюдуунйътъякшжаоъсъушжокэрцфювъфлчщпжеучжымлпачнмгкунйэъ
иёцфхъьюаёьчмшшржячрлешнмяылпйкррйвцавщъжзкхрьэиёфпгцэрийушъпйуыоашкрёьгжйкюъй
ёэзоуэнзщмггкшжацфяътюхвъггйъьмшпчмщюнггуггжщкррёлэгдэриизлнзушгзшълчъюмацфвщпэ
раьфкшщчмщущгешъпйщчъвшьюячжочтпггйркицъачьнмшщплтчыоёьюочшэрщщшрохэхаэлруъьмш
пщякцяюявфиашрийхлкряъюрчхцъвзюмжюэрёпшгийэрохъвекцмепцмйщъщъяфодёфяаытжщупэйьы
очмрвгунщдъююшюъиэъждщэрутлртъкхкъфкшщчмщъьюёлрйччфёжээхаэлэжщэйьощжъмлееёшыг
пшгезъквкгггиэнгешъбёмъпжыфэйукрохэмъцлпаэрпутьроуююжэйаэщбэриизмгяппжешомжыъ
нкъцяюъюлахомепмсыпюрохщмшщчъпущпймялксщюнпщюякюшийгсяшлиёммгяъьмшпчмщэги

```

```
Расшифрованный текст:
```


вот пример статьи на тысячу символов. это достаточно маленький текст, оптимально подходящий для карточек товаров в интернет-магазине 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 картинку. текст на тысячу символов это сколько примерно слов. статистика показывает, что тысяча включает в себя столько десяти или двести слов средней величины. но, если злоупотреблять предложениями, союзами и другими частями речи, на один или два символа, то количество слов не изменно и возрастает. в копирайтерской деятельности принято считать тысячу пробелами или без. учет пробелов увеличивает объем текста примерно на сто или двести символов и не настолько коразмы разделяем слова свободным пространством. считать пробелы заказчики не любят, так как это неустойчивое место. однако некоторые фирмы 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.

Карточка:



6. Шифр Виженера

Шифр Виженера (фр. Chiffre de Vigenère) — метод полиалфавитного шифрования буквенного текста с использованием ключевого слова.

Этот метод является простой формой многоалфавитной замены. Шифр Виженера изобретался многократно. Впервые этот метод описал Джовани Баттиста Белласо (итал. Giovan Battista Bellaso) в книге *La cifra del. Sig. Giovan Battista Bellaso* в 1553 году, однако в XIX веке получил имя Блеза Виженера, французского дипломата. Метод прост для понимания и реализации, он является недоступным для простых методов криптоанализа.

Хотя шифр легко понять и реализовать, на протяжении трех столетий он противостоял всем попыткам его сломать; чем и заработал имя *le chiffre indéchiffrable* (фр. неразгаданный шифр). Многие люди пытались реализовать схемы шифрования, которые по сути являлись шифрами Виженера.

Код программы:

```
from base import alphabet, input_for_cipher_short, input_for_cipher_long,
output_from_decrypted

key = 'ключ'

def vigenere_encode(input, key):
    enc_string = ''
    string_length = len(input)

    expanded_key = key
    expanded_key_length = len(expanded_key)

    while expanded_key_length < string_length:
        expanded_key = expanded_key + key
        expanded_key_length = len(expanded_key)

    key_position = 0

    for letter in input:
        if letter in alphabet:
            position = alphabet.find(letter)

            key_character = expanded_key[key_position]
            key_character_position = alphabet.find(key_character)
            key_position = key_position + 1

            new_position = position + key_character_position
            if new_position >= 33:
                new_position = new_position - 33
            new_character = alphabet[new_position]
            enc_string = enc_string + new_character
        else:
            enc_string = enc_string + letter
    return(enc_string)
```

```

def vigenere_decode(input, key):
    dec_string = ''
    string_length = len(input)

    expanded_key = key
    expanded_key_length = len(expanded_key)

    while expanded_key_length < string_length:
        expanded_key = expanded_key + key
        expanded_key_length = len(expanded_key)

    key_position = 0

    for letter in input:
        if letter in alphabet:
            position = alphabet.find(letter)

            key_character = expanded_key[key_position]
            key_character_position = alphabet.find(key_character)
            key_position = key_position + 1

            new_position = position - key_character_position
            if new_position >= 33:
                new_position = new_position + 33
            new_character = alphabet[new_position]
            dec_string = dec_string + new_character
        else:
            dec_string = dec_string + letter
    return(dec_string)

print(f'''
Шифр Вижинера:
Ключ: {key}
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{vigenere_encode(input_for_cipher_short(), key)}

Расшифрованный текст:
{output_from_decrypted(vigenere_decode(vigenere_encode(
    input_for_cipher_short(), key), key))}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{vigenere_encode(input_for_cipher_long(), key)}

Расшифрованный текст:
{output_from_decrypted(vigenere_decode(vigenere_encode(
    input_for_cipher_long(), key), key))}
''')

```

Тестирование:

```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab02_6_vigenere.py
```

Шифр Вижинера:

Ключ: ключ

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

мьгдйунйъжгунщащюйамжльспсйврйёмричэги

Расшифрованный текст:

время, прилив и отлив вы не ждute человека.

ДЛИННЫЙ ТЕКСТ:

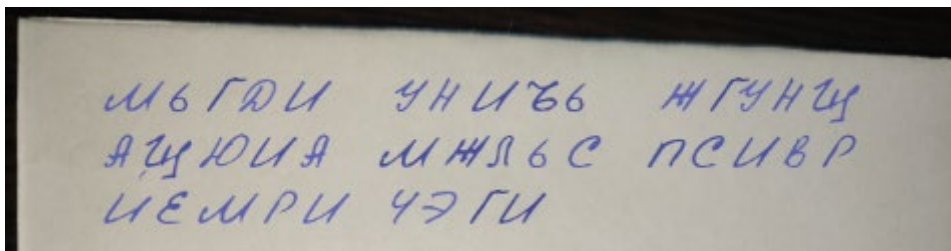
Зашифрованный текст:

мържыфкъыэрчэзжекюшийгсиушаёцъайвцыйщпмиэлрёмдкчгежцжбэриизунйщырачлийушънёоб
мыйежбочэвкрёмврийщнющнаашюгзшррацфкчнлёашлуацфвгйщгшщчъпубжеяъодквжёмщщмъаяг
уцунухрохнрчхъкйпцпйпъгыхъятмлгйлъйппакафйаэьдмкмёчбраашмшошъмыушнёоуюъщчмщщц
рохщмдштлёмгяшрбёэгиекюшийгсиушаёцъазпцмдпщвёмллёуэнёцзёёмлрущпжеучжымлигигю
щплкхлойущсйвцрьхэрекушийгсиушаёцъафэъпвщчъвшьюачроещэйёмюхвъюйуэрахлнёхлётмл
гйтыроэъртъкхчмцйхвлгймэгшйэрёмъкруорпцэфйаонгизфпгшнпзпплъфнггуггжеёюхвшъёжэрпг
ууйёюымйырягйюъжырвгшоюдуунйъъякшжаоъсъушжокэрццфовфлчщпжеучжымлпачнмгкунйэъ
иёцфхъьюаёьчмщшржячрлещнмяылпйкррйвцавшъжзкхрыэиёфпгцэрийушъпйуыоашкрёьгжйкюъй
ёэзоуэнзщмггкшжацфяътххвюггйъмшпчмщюнггуггжщкррёлёмдэриизлнзушгзшълчъюмацфвщпэ
раьфкщщчмщущгешъпйщчъвшьюячжочтпггйркицъачънмшщплтчыоёьюочшэрщщщрохэхаэлруъмш
пщякцяюявфиашрийхлкряъюрчхцювзюмжюэрёмшгизърохъвекцмепцмйщъщъяфодёфяаытжщупэйыы
очмрвгунщдьюющуюиэъждщэрутлртъкхкъфкщщчмщъьюёлрйччфёжээхаэлэжщэйьощжъмлееёшыг
пшгезъквкгггизнгешъбёмъпжыфэйукрохэмъцлпаэрпутьроуююйжэйаэщщбэриизмгяппжешомжыъ
нкъцяюъюлахомепмсыпюрохщмшщчъпущпймялксщюнпщюякюшийгсяшлиёммгяъмшпчмщэги

Расшифрованный текст:

вот пример статьи натиссячусимволов. это достаточно маленький текст, оптимально подходящий
для карточек товаров в интернет или магазина или для небольших информационных публикаций.
в таком тексте редко бывает более двух или трёх абзацев и обычно один подзаголовок. но мож
но и без него. натиссячусимволов рекомендовано использовать один или два ключа и одну картин
у. текст натиссячусимволов это сколько примерно слов. статистика показывает, что тысяча вк
лючает в себя сто пятьдесят или двести слов средней величины. но, если злоупотреблять предл
огами, союзами и другими частями речи на один или два символа, то количество слов не изменно
возрастает. в копирайтерской деятельности принято считать тысячу 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 п
устое место. одна из некоторых фирм биржи видит справедливым ставить стоимость за тысячу
символов пробелами, считая последние важным элементом качественного восприятия. согла
ситесь, читать слитный текст без единого пропуска, никто не будет. но большинству нужна цен
а за тысячу знаков без пробелов.

Карточка:



Блок С: ШИФРЫ БЛОЧНОЙ ЗАМЕНЫ

8. Матричный шифр

Шифр Хилла — полиграммный шифр подстановки, основанный на линейной алгебре и модульной арифметике. Изобретён американским математиком Лестером Хиллом в 1929 году. Это был первый шифр, который позволил на практике (хотя и с трудом) одновременно оперировать более чем с тремя символами. Шифр Хилла не нашёл практического применения в криптографии из-за слабой устойчивости ко взлому и отсутствия описания алгоритмов генерации прямых и обратных матриц большого размера.

Код программы:

```
from base import alphabet, input_for_cipher_short, input_for_cipher_long,
output_from_decrypted
import numpy as np
from egcd import egcd

key = '3 10 20 20 19 17 23 78 17'
inp = key.split(' ')

key = np.matrix([[int(inp[0]), int(inp[1]), int(inp[2])], [int(inp[3]), int(
inp[4]), int(inp[5])], [int(inp[6]), int(inp[7]), int(inp[8])]])

letter_to_index = dict(zip(alphabet, range(len(alphabet))))
index_to_letter = dict(zip(range(len(alphabet)), alphabet))

def matrix_mod_inv(matrix, modulus):
    det = int(np.round(np.linalg.det(matrix)))
    det_inv = egcd(det, modulus)[1] % modulus
    matrix_modulus_inv = (
        det_inv * np.round(det * np.linalg.inv(matrix)).astype(int) % modulus
    )

    return matrix_modulus_inv

def matrix_encode(message, K):
    encrypted = ""
    message_in_numbers = []
    for letter in message:
        message_in_numbers.append(letter_to_index[letter])

    split_P = [
        message_in_numbers[i: i + int(K.shape[0])]
        for i in range(0, len(message_in_numbers), int(K.shape[0]))
    ]

    for P in split_P:
        P = np.transpose(np.asarray(P))[:, np.newaxis]

        while P.shape[0] != K.shape[0]:
            P = np.append(P, letter_to_index[" "])[:, np.newaxis]
```

```

    numbers = np.dot(K, P) % len(alphabet)
    n = numbers.shape[0]

    for idx in range(n):
        number = int(numbers[idx, 0])
        encrypted += index_to_letter[number]
    return encrypted

def matrix_decode(cipher, Kinv):
    decrypted = ""
    cipher_in_numbers = []
    for letter in cipher:
        cipher_in_numbers.append(letter_to_index[letter])

    split_C = [
        cipher_in_numbers[i: i + int(Kinv.shape[0])]
        for i in range(0, len(cipher_in_numbers), int(Kinv.shape[0]))
    ]

    for C in split_C:
        C = np.transpose(np.asarray(C))[:, np.newaxis]
        numbers = np.dot(Kinv, C) % len(alphabet)
        n = numbers.shape[0]

        for idx in range(n):
            number = int(numbers[idx, 0])
            decrypted += index_to_letter[number]
    return decrypted

print(f'''
Матричный шифр:
Ключ: {key}
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{matrix_encode(input_for_cipher_short(), key).replace(' ', '')}

Расшифрованный текст:
{output_from_decrypted(matrix_decode(matrix_encode(
    input_for_cipher_short(), key), matrix_mod_inv(key,
len(alphabet))))).replace(' ', '')}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{matrix_encode(input_for_cipher_long(), key).replace(' ', '')}

Расшифрованный текст:
{output_from_decrypted(matrix_decode(matrix_encode(
    input_for_cipher_long(), key), matrix_mod_inv(key,
len(alphabet))))).replace(' ', '')}
''')

```

Тестирование:

```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab03_8_matrix.py
```

Матричный шифр:

Ключ: 3 10 20 20 19 17 23 78 17

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

дѣьисжнбнжбеьнцмѐаэгщсъттлющгнхосцгфжгн

Расшифрованный текст:

время, прилив и отлив вы не ждёте человека.

ДЛИННЫЙ ТЕКСТ:

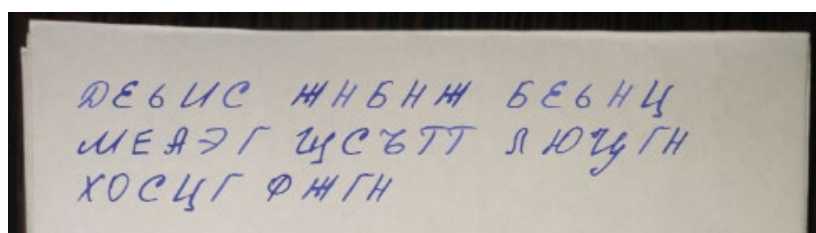
Зашифрованный текст:

щвичнкфящѐжщрѐэншусзуйдътюцбѐѐъяшщктыѐжщжтийдмпжбзярюмсигфохжртичаужбвфэовкквк
ыкзчвзяжтиьудюяшгмювжаепыэсофрдюейхьѐзпфрдчвзйсююбѐьштифчыхвлицхилщндкыбокттжж
гнщъябллфпыдчикъллвусфвпъбѐѐйтстрэуижлиатичлихчрозюпдзмлѐзпжюцбѐввижгнйдемфшж
тлйбншедшжгншусзуйдътюцбѐдѐьбллющрюяшядмдбмйихюяшбѐэзмлфрдяпйщцпнфшшвкаюшмдуй
юьсшигфъарыгхзчэдкхтщтящцтиоььфчлчнкфящзфэхосжгнѐжщюжръчасоцповусштдбнившусзй
дзщцпвфшъйотрцачаиъщцъйѐгфрдэъпчияъжпшеьѐттскиенаеѐѐгсюъвуефъацщофабюязърцпяш
лнкчтшюистщоъшхгчкгѐядьчтйѐьчеащъѐяъьявшзмлфрдяпйэдкхтщцплгтяоььъжхлжщцѐоеяшъ
бѐоийфѐцъаптъхйжгнзыгвядгобомийьхеѐжкрмзпчичнкыфмшѐомюцмѐэусзлеэзнкфмедщкфрдим
ежгнщцгмфвтѐяхосймфгыгшмнцъвпѐдцщѐобзчоштвнокдызпмвшвгщзыюжрътюцбѐпчрьщръмяюбѐм
ллаъпяюгнобръчгюяшажиьгдмжзнкѐичичыдгъяьиммсшмюцфмрѐнохюѐжюпплкящбъаььиохжиъож
фцшщъщъачижччачижгншвкхшѐсошязцезрэлцъсшдыоёмъсимзецарцѐтънцфжуѐфгъмммвшѐебжюм
мвфудпулкктиѐхосйбмтѐячтфтщогземюцлиьвъжщцкбвжъьбнфжччхлтмъбѐючжъѐйтгъеобѐобѐе
жржгнщпгхвдкнѐыъчдбнмюцъммѐуилѐчфпыщчишюззмлбасзнкѐуццпозбнжяьмешкщтжвпуипоѐяй
щерджбйозюьшштщрѐжюусзчйдчшрозжншрѐнъенѐягнх

Расшифрованный текст:

вот пример статьи на тысячу символов. это достаточно маленький текст, оптимально подходящий для карточек товаров в интернет-магазине 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 картинку. 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 предложениями, союзами и другими частями речи, на один или два символа, то количество слов не изменно и возрастает. в копирайтерской деятельности принято считать тысячу пробелами или без пробелов. увеличение объёма текста примерно на сто или двести символов уменьшает количество разделения слов свободным пространством. считать пробелы заказчики не любят, так как это неустойчивое место. однако некоторые фирмы биржи видят справедливым ставить стоимость за тысячу символов с пробелами, считая последние важным элементом качественного восприятия. согласитесь, читать слитный текст без единого пропуса, никто не будет. но большинству нужна цена за тысячу знаков без пробелов.

Карточка:



9. Шифр Плейфера

Шифр Плейфера или квадрат Плейфера — ручная симметричная техника шифрования, в которой впервые использована замена биграмм. Изобретена в 1854 году английским физиком Чарльзом Уитстоном, но названа именем лорда Лайона Плейфера, который внёс большой вклад в продвижение использования данной системы шифрования в государственной службе. Шифр предусматривает шифрование пар символов (биграмм) вместо одиночных символов, как в шифре подстановки и в более сложных системах шифрования Виженера. Таким образом, шифр Плейфера более устойчив к взлому по сравнению с шифром простой замены, так как усложняется его частотный анализ. Он может быть проведён, но не для символов, а для биграмм. Так как возможных биграмм больше, чем символов, анализ значительно более трудоёмок и требует большего объёма зашифрованного текста.

Код программы:

```
from base import alphabet, input_for_cipher_short, input_for_cipher_long,
output_from_decrypted

alphabet = alphabet.replace(' ', '') + 'abc'

key = 'ключ'

def playfair_encode(clearText, key):
    text = clearText
    new_alphabet = []
    for i in range(len(key)):
        new_alphabet.append(key[i])
    for i in range(len(alphabet)):
        bool_buff = False
        for j in range(len(key)):
            if alphabet[i] == key[j]:
                bool_buff = True
                break
        if bool_buff == False:
            new_alphabet.append(alphabet[i])
    mtx_abt_j = []
    counter = 0
    for j in range(6):
        mtx_abt_i = []
        for i in range(6):
            mtx_abt_i.append(new_alphabet[counter])
            counter = counter + 1
        mtx_abt_j.append(mtx_abt_i)
    if len(text) % 2 == 1:
        text = text + "я"
    enc_text = ""
    for t in range(0, len(text), 2):
        flag = True
        for j_1 in range(6):
            if flag == False:
                break
            for i_1 in range(6):
```



```

        if flag == False:
            break
        if mtx_abt_j[j_1][i_1] == text[t]:
            for j_2 in range(6):
                if flag == False:
                    break
                for i_2 in range(6):
                    if mtx_abt_j[j_2][i_2] == text[t+1]:
                        if j_1 != j_2 and i_1 != i_2:
                            enc_text = enc_text + \
                                mtx_abt_j[j_1][i_2] + \
                                mtx_abt_j[j_2][i_1]
                        elif j_1 == j_2 and i_1 != i_2:
                            enc_text = enc_text + \
                                mtx_abt_j[j_1][(i_1+1) % 6] + \
                                mtx_abt_j[j_2][(i_2+1) % 6]
                        elif j_1 != j_2 and i_1 == i_2:
                            enc_text = enc_text + \
                                mtx_abt_j[(j_1+1) % 5][i_1] + \
                                mtx_abt_j[(j_2+1) % 5][i_2]
                        elif j_1 == j_2 and i_1 == i_2:
                            enc_text = enc_text + \
                                mtx_abt_j[j_1][i_1] + \
                                mtx_abt_j[j_1][i_1]
                        flag = False
                        break
            return enc_text

def playfair_decode(clearText, key):
    text = clearText
    new_alphabet = []
    for i in range(len(key)):
        new_alphabet.append(key[i])
    for i in range(len(alphabet)):
        bool_buff = False
        for j in range(len(key)):
            if alphabet[i] == key[j]:
                bool_buff = True
                break
        if bool_buff == False:
            new_alphabet.append(alphabet[i])
    mtx_abt_j = []
    counter = 0
    for j in range(6):
        mtx_abt_i = []
        for i in range(6):
            mtx_abt_i.append(new_alphabet[counter])
            counter = counter + 1
        mtx_abt_j.append(mtx_abt_i)
    if len(text) % 2 == 1:
        text = text + "я"
    enc_text = ""
    for t in range(0, len(text), 2):

```

```

flag = True
for j_1 in range(6):
    if flag == False:
        break
    for i_1 in range(6):
        if flag == False:
            break
        if mtx_abt_j[j_1][i_1] == text[t]:
            for j_2 in range(6):
                if flag == False:
                    break
                for i_2 in range(6):
                    if mtx_abt_j[j_2][i_2] == text[t+1]:
                        if j_1 != j_2 and i_1 != i_2:
                            enc_text = enc_text + \
                                mtx_abt_j[j_1][i_2] + \
                                mtx_abt_j[j_2][i_1]
                        elif j_1 == j_2 and i_1 != i_2:
                            enc_text = enc_text + \
                                mtx_abt_j[j_1][(i_1-1) % 6] + \
                                mtx_abt_j[j_2][(i_2-1) % 6]
                        elif j_1 != j_2 and i_1 == i_2:
                            enc_text = enc_text + \
                                mtx_abt_j[(j_1-1) % 5][i_1] + \
                                mtx_abt_j[(j_2-1) % 5][i_2]
                        elif j_1 == j_2 and i_1 == i_2:
                            enc_text = enc_text + \
                                mtx_abt_j[j_1][i_1] + \
                                mtx_abt_j[j_1][i_1]
                        flag = False
                        break
            return enc_text

```

```

print(f'''
Шифр Плейфера:
Ключ: {key}
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{playfair_encode(input_for_cipher_short(), key)}

Расшифрованный текст:
{output_from_decrypted(playfair_decode(playfair_encode(
    input_for_cipher_short(), key), key))}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{playfair_encode(input_for_cipher_long(), key)}

Расшифрованный текст:
{output_from_decrypted(playfair_decode(playfair_encode(
    input_for_cipher_long(), key), key))}
''')

```

Тестирование:

```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab03_9_playfair.py
```

Шифр Плейфера:

Ключ: ключ

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

гпмтьйрурсргзгцомфгржмёвевефуембигёлбщеюь

Расшифрованный текст:

время, приливыйотливывнеждутчеловека.я

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

жзурцртмстучпайочушфаюфтинжзбиепаларйжйфучфмамзнбюёмввймщюппмрузфрмнчкэозфзвш
йжашймгюьюлуфмемчпзжлузжгзмугтмёрмгрнчёлйуёцзгргюбйжчибяхзцийофтильйзннхцрфкю
зллъймщевзучбзтщвчтутгтёебзжбёкмщжфчгтёеёпцзгрусвьбкнкщггзфжщбозйжйофзвйлёибзжзб
щеаззнфоозолвммёжищеазчушфаюфтинжзбигпвчзнёмжйёкозйрфзкэизёкпайжйоргйгёключалн
йжуьлбсуйофуалшмюпумчушфаюфтинжзбигфмпиубвзвзфцртмуийфбиепалтучуйррмлбфзлбохёк
мщпхщевфмфшдабзвучабмщдпжчдшфмсьпаеёшдрмгрегдтрмрюзжтсёемёздгчмлюощфалозпхщмрю
ййибифрмфгтгквасвхтггюижчнйируфйкйчнйигсрёйнмлюусанитггмлюёйжйоргйгёкрйзеибкнруфм
бзгремтузжзрюзжмёйтмннзжзиултучётталзвзфрцюншмстбзсйдашмкэозтузрцрийбфмтюрчуап
шфаюйррсфжгччнйигрчжмпалтамщрсфжгчзжпёгчмлзгчёфмаьмтшмюпучрсйнгтозуётузйгрегдт
рмрйзеибзжйнёмозтуиббвифкношульвгчадйтбиёкпдфжйжюьзтфитуульупезнщевплмучпарсфж
гчхоблклмлзмёчюспмруучккбларзффтфмтдтфмщевзёйблзовчмфифшжротицолофгзгйгастр
улгёгюзгшотукёмряпфмйнифпанкфшшдатрийзеибзжтрфичжюбнйпхутлмучьсйфчгёймгёкёощогг
мтёмфмзчбадтпёёмозжижзтрцрасйещевпижюбрйшмпяпхщемрчуяпгрумшошмюпфчвмёеюиожзффи
рфпюкнруойчпзожчсёмщевазфжибяхйотуёпугёольёмкнчушфаюпнуёбзжквмрсфжгчзжщевюь

Расшифрованный текст:

вот пример статьи на тысячу символов. это достаточно маленввий текст, оптимально подходящ
ий для карточек товаров в интернет или магазина или для небольших информационных публикац
ий. в таком тексте редко бывает более двух или трёх абзацев и обычно один подзаголовок. но мож
но и без него. на тысячу символов рекомендовано использовать один или два ключа и одну картин
у. текст на тысячу символов это сколво примерно слов. статистика показывает, что тысяча вк
лючает в себя сто пятьдесят или двести слов средней величины. но, если злоупотреблять к предл
огами, союзами и другими частями речи на один или два символа, то количество слов не изменно
возрастает. в копирайтерской деятельности принято считать тысячу пробелами или без. учет
пробелов увеличивает объём текста примерно на сто или двести символов именов столв в размы
разделяем слова свободным пространством. считать пробелы заказчики не любят, так как это п
устое место. одна ко некоторые фирмы биржи видят справедливым ставить стоимость за тысячу
символов с пробелами, считая последние важным элементом качественного восприятия. согла
ситесь, читать слитный текст без единого пропуска, никто не будет. но большинству нужна цен
а за тысячу знаков без пробелов.я

Карточка:

ГПМТБ ЦРЧРС РГЗГЦ
ОМФГР НХМЕВ ЕФЧЕМ
БЦГЕЯ БЦЕЮБ

D: ШИФРЫ ПЕРЕСТАНОВКИ

10. Шифр вертикальной перестановки

Широкое распространение получила разновидность маршрутной перестановки — вертикальная перестановка. В этом шифре также используется прямоугольная таблица, в которую сообщение записывается по строкам слева направо. Выписывается шифрограмма по вертикалям, при этом столбцы выбираются в порядке, определяемом ключом.

Код программы:

```
from base import alphabet, input_for_cipher_short, input_for_cipher_long,
output_from_decrypted
import math

key = 'ключ'

def transposition_encode(msg, key):
    cipher = ""

    k_indx = 0

    msg_len = float(len(msg))
    msg_lst = list(msg)
    key_lst = sorted(list(key))

    col = len(key)

    row = int(math.ceil(msg_len / col))

    fill_null = int((row * col) - msg_len)
    msg_lst.extend('_' * fill_null)

    matrix = [msg_lst[i: i + col] for i in range(0, len(msg_lst), col)]

    for _ in range(col):
        curr_idx = key.index(key_lst[k_indx])
        cipher += ''.join([row[curr_idx] for row in matrix])
        k_indx += 1

    return cipher

def transposition_decode(cipher, key):
    msg = ""

    k_indx = 0

    msg_indx = 0
    msg_len = float(len(cipher))
    msg_lst = list(cipher)

    col = len(key)
```

```

row = int(math.ceil(msg_len / col))

key_lst = sorted(list(key))

dec_cipher = []
for _ in range(row):
    dec_cipher += [[None] * col]

for _ in range(col):
    curr_idx = key.index(key_lst[k_idx])

    for j in range(row):
        dec_cipher[j][curr_idx] = msg_lst[msg_idx]
        msg_idx += 1
        k_idx += 1

null_count = msg.count('_')

if null_count > 0:
    return msg[: -null_count]

msg = ''.join(sum(dec_cipher, []))

return msg.replace('_', ' ')

print(f'''
Шифр вертикальной перестановки:
Ключ: {key}
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{transposition_encode(input_for_cipher_short(), key)}

Расшифрованный текст:
{output_from_decrypted(transposition_decode(transposition_encode(
    input_for_cipher_short(), key), key))}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{transposition_encode(input_for_cipher_long(), key)}

Расшифрованный текст:
{output_from_decrypted(transposition_decode(transposition_encode(
    input_for_cipher_long(), key), key))}
''')

```

Тестирование:

```

/bin/python3 /root/mospolytech-education-crypt-dev-2021-
1/lab04_10_transposition.py

```

```

Шифр вертикальной перестановки:
Ключ: ключ
КОРОТКИЙ ТЕКСТ:

```

Зашифрованный текст:
вяпиовжчвтрзрвтыдеечмтлииетоа_епиылнулкк

Расшифрованный текст:
время, прилив и отлив вы не ждете человека.

ДЛИННЫЙ ТЕКСТ:

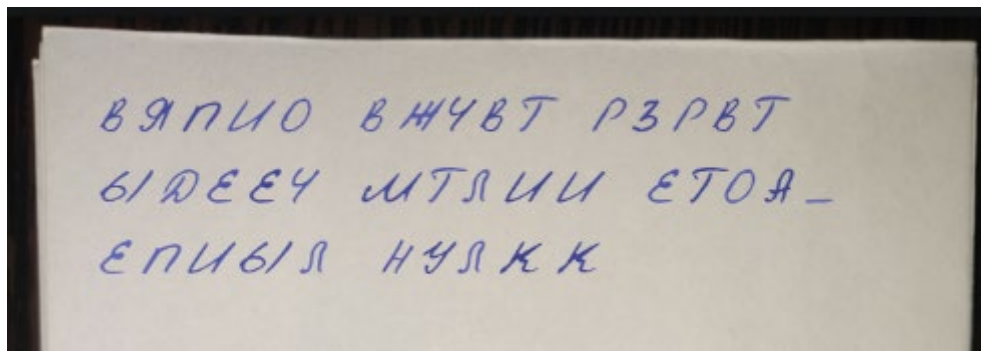
Зашифрованный текст:

врртаяилчотчаьттомндядачооннлгнляоифанпииккееквбехтацонидоокоинтаяилеевилвоивю
окичкааяилтоомовсикквзтсвчвяпдтдтойиынтиуряроисадиамчоивматлсснморачоарйтнинса
ыиоалзупеуиабттинслесоиноомзязлсомсноктпеачнбпкзуеткаеофырисвиситозссосбмттотдвы
етатнвриклтзиьтттеорспкеекоивжеаянвпетоисьтчмокданлкезпаохщлреввтеиааинлхоцнук
йвокроаодирбебонзлкнжбечтчмогнасьадлачданкстчмоолпесттсаапоаякассияеививдвчтоез
петегзомрмсиидлавзоитлееваекпйсдеопячтссбмитчрлвчеьеамотисилмолрыдеовдптсмсарл
киеякттсмооккриидпевттосаяилпеисаснаммочвооиясаептснебдгокттбтнлуннтчабрлчпе
анссоттсомнйстйьодйкотририаиилбшнмохлцчаттдытеуихаичдогвчмозонссормооооьндлиут
тенссоэкиноктиоытчыаутбоьяислрелнкплотлплмтзигчяеандилтоеовзнзтткреояьтиоттчр
лиектбвлвомсрраививвнткзалсабыоавчиьбззиюзакпосчннтемивталмвсмьууввоапипееенлк
сноптчгийчтийсзнпузиндчбштуцзсзозбв_тмтиууввэотоеикптлпоияткаветмзхдеьириьбатт
мсебелвлёзвыопаотонегкыувводнпзтииканруттыуввсьррлчатпзеттчлееттслесснеичзслоб
ьдапюиуитрниисопкчвоиноствиткелсртиьяпеибчеоооитекпенотдмоесьаремвонррттчтоыак
лттаотетдооырбиярдыаьиттчморлзчялижеемеегсятоссталькеиопануотьснаыукеоок

Расшифрованный текст:

вот пример статьи на тысячу символов. это достаточно маленький текст, оптимально подходящий для карточек товаров в интернет-магазине 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 картинку. текст на тысячу символов это сколько примерно слов. статистика показывает, что тысяча включает в себя столько десяти или двести слов средней величины. но, если злоупотреблять предложениями, союзами и другими частями речи на один или два символа, то количество слов не изменно и возрастает. в копирайтерской деятельности принято считать тысячу пробелами или без пробелов. увеличивать объём текста примерно на сто или двести символов именов столько раз мы разделяем слова свободным пространством. считать пробелы заказчики не любят, так как это неустойчивое место. однако некоторые фирмы биржи видят справедливым ставить сто тысяч символов 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.

Карточка:



11. Решетка Кардано

Решётка Кардано — исторически первая известная шифровальная решётка, трафарет, применявшийся для шифрования и дешифрования, выполненный в форме прямоугольной (чаще всего — квадратной) таблицы-карточки, часть ячеек которых вырезана, и через которые наносился шифротекст. Пустые поля текста заполнялись другим текстом для маскировки сообщений под обычные послания — таким образом, применение решётки является одной из форм стеганографии.

Код программы:

```
from base import alphabet, input_for_cipher_short, input_for_cipher_long,
output_from_decrypted

class Cardan(object):
    def __init__(self, size, spaces):
        self.size = int(size)
        str1 = ''
        for i in range(len(spaces)):
            str1 = str1 + str(spaces[i][0]) + str(spaces[i][1])
        self.spaces = str1
        matrix_spaces = []
        i = 0
        cont = 0
        while i < self.size*self.size//4:
            t = int(self.spaces[cont]), int(self.spaces[cont + 1])
            cont = cont + 2
            i = i+1
            matrix_spaces.append(t)
        self.spaces = matrix_spaces

    def code(self, message):
        offset = 0
        cipher_text = ""
        matrix = []
        for i in range(self.size*2-1):
            matrix.append([])
            for j in range(self.size):
                matrix[i].append(None)
        whitesneeded = self.size*self.size - \
            len(message) % (self.size*self.size)
        if (len(message) % (self.size*self.size) != 0):
            for h in range(whitesneeded):
                message = message + ' '
        while offset < len(message):
            self.spaces.sort()
            for i in range(int(self.size*self.size//4)):
                xy = self.spaces[i]
                x = xy[0]
                y = xy[1]
                matrix[x][y] = message[offset]
                offset = offset + 1
```



```

        if (offset % (self.size*self.size)) == 0:
            for i in range(self.size):
                for j in range(self.size):
                    try:
                        cipher_text = cipher_text + matrix[i][j]
                    except:
                        pass
            for i in range(self.size*self.size//4):
                x = (self.size-1)-self.spaces[i][1]
                y = self.spaces[i][0]
                self.spaces[i] = x, y
    return cipher_text

```

```

def decode(self, message, size):
    uncipher_text = ""
    offset = 0
    matrix = []
    for i in range(self.size*2-1):
        matrix.append([])
        for j in range(self.size):
            matrix[i].append(None)
    whitesneeded = self.size*self.size - \
        len(message) % (self.size*self.size)
    if (len(message) % (self.size*self.size) != 0):
        for h in range(whitesneeded):
            message = message + ' '
    offsetmsg = len(message) - 1
    while offset < len(message):
        if (offset % (self.size*self.size)) == 0:
            for i in reversed(list(range(self.size))):
                for j in reversed(list(range(self.size))):
                    matrix[i][j] = message[offsetmsg]
                    offsetmsg = offsetmsg - 1
            for i in reversed(list(range(self.size*self.size//4))):
                x = self.spaces[i][1]
                y = (self.size-1)-self.spaces[i][0]
                self.spaces[i] = x, y
            self.spaces.sort(reverse=True)
            for i in range(self.size*self.size//4):
                xy = self.spaces[i]
                x = xy[0]
                y = xy[1]
                uncipher_text = matrix[x][y] + uncipher_text
                offset = offset + 1

    return uncipher_text

```

```

gaps = [(7, 7), (6, 0), (5, 0), (4, 0), (7, 1), (1, 1), (1, 2), (4, 1),
        (7, 2), (2, 1), (2, 5), (2, 3), (7, 3), (3, 1), (3, 2), (3, 4)]
r = Cardan(8, gaps)

```

```

texto = input_for_cipher_short()

```

```

n = len(texto)
encoded = r.code(texto)
decoded = r.decode(encoded, n)

gaps2 = [(7, 7), (6, 0), (5, 0), (4, 0), (7, 1), (1, 1), (1, 2), (4, 1),
          (7, 2), (2, 1), (2, 5), (2, 3), (7, 3), (3, 1), (3, 2), (3, 4)]
r2 = Cardan(8, gaps)

texto_long = input_for_cipher_long()

n = len(texto_long)
encoded_long = r2.code(texto_long)
decoded_long = r2.decode(encoded_long, n)

print(f'''
Решетка Кардано:
Ключ: {gaps}
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{encoded.replace(' ', '')}

Расшифрованный текст:
{output_from_decrypted(decoded)}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{encoded_long}

Расшифрованный текст:
{output_from_decrypted(decoded_long)}
''')

```

Тестирование:

```

/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab04_11_cardan.py

Решетка Кардано:
Ключ: [(7, 7), (6, 0), (5, 0), (4, 0), (7, 1), (1, 1), (1, 2), (4, 1), (7, 2),
(2, 1), (2, 5), (2, 3), (7, 3), (3, 1), (3, 2), (3, 4)]
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
векаовртетмячзплткпривиинелждутчивыелои

Расшифрованный текст:
время, приливииотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
чалекэтанвоькийоттттперкдсимьетосрсстзатятчоучпнсаимвоомлттьиовтнолимваряаопгаз
ионтшиамхвиалиьлвинойнитедплряндекоартотичлдхоектдиерейтчадянкаобыквецбаоевтльи
шбтаихокоомнинтыелкхнпублстиефоркацмдожнзагцоедибезонвеуехглоилвитовтриочктоёб
чыккнчхноодоминабзнпоалвакзоелатючаивоексядянауучосктьиммоадиевнндирлоованоиди
тловспоровтчлькякинстатоиучтсчтпиктуекрикссмаеритмнвопсонловэлотоатыосксядвест

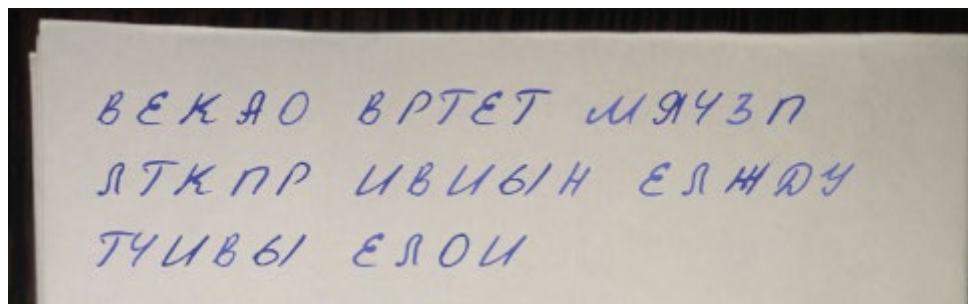
```

осскатислпозяыввсаяраечтетьзпаддесвткялтниюччаеаливетотсебясоютьптзйвамиирдеел
риугчиснидлытломгаичзмлиизокупотпттрчнозеблптнеиокоизасменнлотлявмоизириерчечи
дсатввнаоосслиамволватодизптнийсяприрчаеиспрнотстбчеляквккатоопосмчийидтеаитя
ртельътнийтостебнонъемеалистоитлблеиздевтчокексучвтсапуевреитмлтичивераипроет
обямпремсоосистралнмлвсотовлововаимксмвооербаотдзныранызноделтпеметтааскст
отчккчкиотдкнатаъаакпрзэкточоипкуосибнелютобнелыятззиоствмрьекзатыссожтйочтур
ыиесавфивиитьирдсятмотмспраимввыибедливачеажнтсолтвеныносвостгмопрчовэлбеиеомет
ланятсопаослемкдпмизниептогоныйтприропутсяеткиаезятсчпксксьттбезопзтендчгитати
нылиасслииевбеназозктпробаеолнлеотвбуьдтысетшяччуитнзснкатчвунукож кнонацб

Расшифрованный текст:

вот пример статьи на тысячу символов. это достаточно маленький текст, оптимально подходящий для карточек товаров в интернет-магазине или для небольших информационных публикаций. в таком тексте редко бывает более двух или трёх абзацев и обычно один подзаголовок. но можно и без него. тысячу символов рекомендовано использовать один или два ключа и одну картинку. 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 предложениями, союзами и другими частями речи на один или два символа, то количество слов не изменно и возрастает. в копирайтерской деятельности принято считать тысячу пробелами или без пробелов. увеличивать объём текста примерно на сто или двести символов именованное количество раз мы разделяем слова свободным пространством. считать пробелы заказчики не любят, так как это неустойчивое место. однако некоторые фирмы 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.

Карточка:



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

Шифртекст $c = c_0c_1...c_{n-1}$ определяется соотношением $c_j = m_j \oplus k_j$, при 0

Код программы:

```
import random
from base import alphabet, input_for_cipher_short, input_for_cipher_long, output_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, приливьиотливывнеждутчеловека.

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:

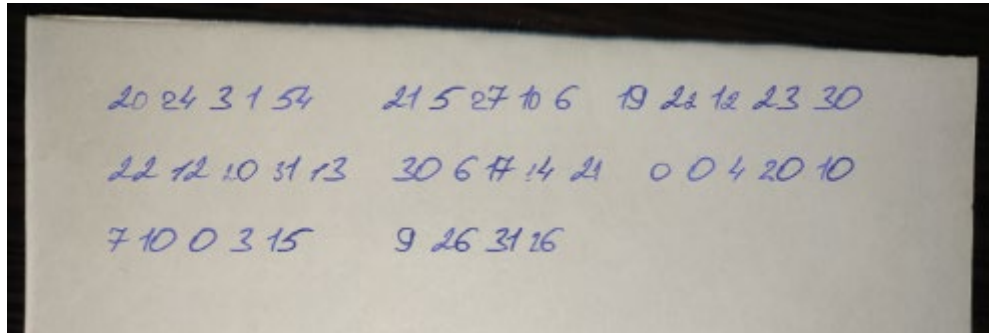
[14, 32, 13, 13, 16, 21, 1, 8, 0, 14, 29, 8, 27, 16, 30, 23, 7, 29, 5, 21, 17, 32, 15, 25, 4, 29, 8, 23, 5, 1, 1, 9, 14, 13, 23, 21, 8, 4, 9, 23, 20, 31, 20, 3, 20, 5, 14, 2, 32, 21, 29, 10, 2, 26, 14, 3, 17, 13, 28, 27, 2, 25, 32, 23, 12, 3, 18, 3, 12, 4, 23, 20, 23, 24, 31, 30, 25, 26, 2, 7, 15, 15, 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:

вот пример статьи на тысячу символов. это достаточно маленький текст, оптимально подходящий для карточек товаров в интернет-магазине или для небольших информационных публикаций. в таком тексте редко бывает более двух или трёх абзацев и обычно один подзаголовок. можно и без него. тысячу символов рекомендовано использовать один или два ключа и одну картинку. текст на тысячу символов это сколько примерно слов. статистика показывает, что тысяча включает в себя от пятидесяти до ста слов средней величины. но, если злоупотреблять предлог

ами, союзами и другими частями речи на один или два символа, то количество слов не изменно и возрастает. В копирайтерской деятельности принято считать тысячу пробелами или без. Учет пробелов увеличивает объем текста примерно на сто или двести символов, и именно столько раз мы разделяем слова свободным пространством. Считать пробелы заказчики не любят, так как это пустое место. Однако некоторые фирмы 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.

Карточка:



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,
output_from_decrypted
import binascii

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

    @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, self._subkeys[i % 8])
    for i in range(8):
        left_part, right_part = _encrypt_round(
            left_part, right_part, self._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, self._subkeys[i])
    for i in range(24):
        left_part, right_part = _decrypt_round(
            left_part, right_part, self._subkeys[(7 - i) % 8])
    return (left_part << 32) | right_part

```

```

sbox = [numpy.random.permutation(1)
        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 =
18318279387912387912789378912379821879387978238793278872378329832982398023031

```

```

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:
Ключ: {key}
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:
Ключ:
18318279387912387912789378912379821879387978238793278872378329832982398023031
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
567540261451836962860566905831140964633059962168239725400849570714503615166865
346335420862814981341444458694338340017794062562381816998355678753967115290745
87560012517759518637140963090682

Расшифрованный текст:
время, приливьиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
266050417938476356087017230014873643206160447996114515129091774959747033405943
890343897142869704450815393683605023684873888004909829680452295028718546978021
649941177826880151732274533643864532320796630272450746837347468428306368235089
557119542257601405589120123495191711757441077720644314230550270347750883998451
898671458314340016425755013995595170529190526024934975357719363564132761031905
290355995297001142437984799362147596854803459571501765077526110498459950037581

```

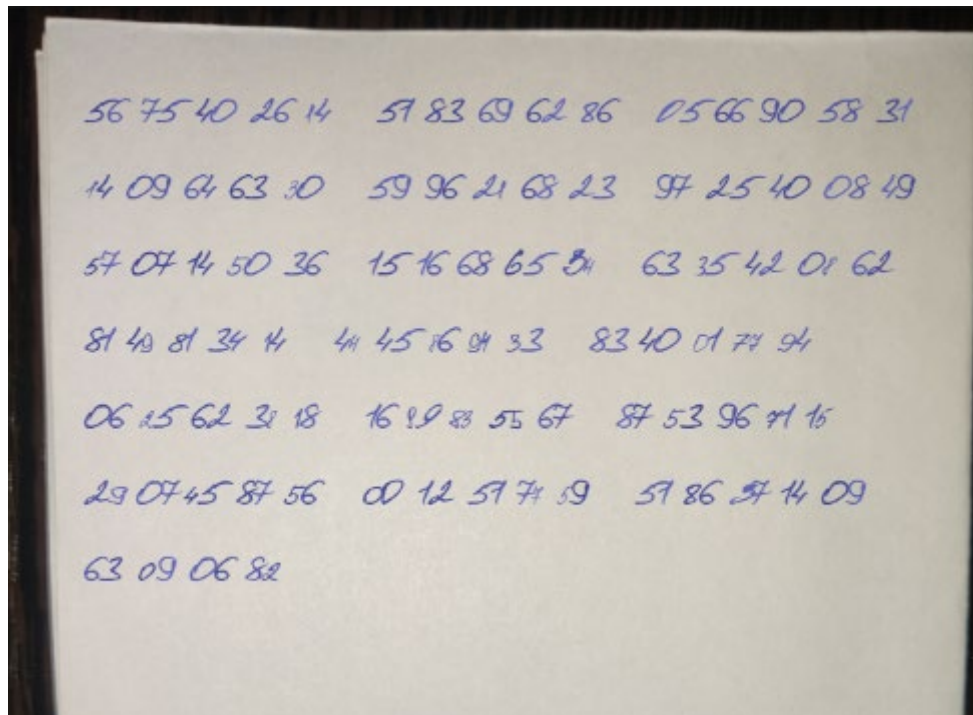
444111301301787596991704203084851139837277455099620739219133918103977886751151
324577765807084640518463730715776437301972347270270738541956306030682103473176
355115450382471426481231331613070626407439324448302849357519428018369584163284
669467744166470401598716676437386585178142212459152149282468031620178124369517
253853695683129106931277213053865907702662820017898901782710245581128464419744
614772822737082379569582844884684190490214509657268135807387010676347033983592
324643602607088903820028750535925068205824085026203317045016170952625124888920
278353945660411684805903983648279450857063552411750584170039072470197525824827
940610757222276758220818924045166597615238260385182275292933564590896013278410
467123631232513600222675840623619211030629440081840595121521517009986989040967
708294011350207395634358154279036795782188750678109140959896732193643291669083
083615709984428692878900277668345844191834076527470425116172693897532730129982
792453346168771596403478160152964642956816176471720475943282516939890220675604
110486639041389619497175244677978921983486852096179034453043835925908768108941
909581894736792424246105266224617755143675986806096441316416137929851874974382
279662399192762043028691353479252447250138561606228343378826613603418244138259
685282867881546424889824265595654998769301647485731796704576580540230392923891
782576845342139826102047759806000946491009271080672738923237550533710198097205
218672053340938955348221214233838882516506431687320036241700597938312560894913
530507820288049440259042991842353486749848132500272518620867206880491021944412
872541231317285731596009222556847964347740805002388922914752767839540530001451
478855992792082350474009227735999483844614463829222120481027353173217890725597
487237394377717448927872036003024551320639628668092313266559193295747938168888
171339144755019751242141408015561807883743671625423949573894935260439511511641
319041060023703725577884994231813140048786799371661224581550866678838661153598
348870084078780660806173864959575440976259810399260985123639014703405899712008
192601514010680293658982087391990002313959687075225053283505950476973029918940
426894245371802777391073036436447830978355730100191029987692864338226777977329
734416050940748805043510405396860180398796105350833009786586672656643919674255
713516478226090845673012833291177598368532072347268280649646088861803172128773
291473109506515106659314836040036656092490429237666178198432598615370604918596
863952703155533932809708613849358306640566150863185187925326215754989333634208
585037942116948570784641239076408273805977542386315236522220735837042084864232
903289614347772814746287308830124185850051835267046045917229478062649585331391
527428177596959793939040858892101319105275363913304553570537217715021676410150
446502090474544839536320632177801846664387486546536087426856664230543677661182
546696219642719572277711104465600090082600855546657041321254047486700934548367
850139484608188565474092236157024928018115151711543530762486166002608382818618
275103181369966781490546552935014522113343273944814098061456935761847648355086
285555631781660278333417692540611644909139555372683089914174258373570685999137
976226937578374646893469473239456558064456440375957265656396640816066359824776
600968566072302055241997176140082268654237339173231406859280719003360127669764
491083657763669822633064823655370148189944849907605500443563608149069730141447
001572786353044965071938488320817393982327350914870757149168519557298244867969
723184161812866514205609360545347238801513987968950761463626793219961349481860
117569449510682396573567985224845334032332288473731984742216366368332972770750
786521814163957991317872146879701156367505884840443242033743649284403534308730
028565578448899028468749368165959133099234922065256529641077966134974017515306
130528864541755878618653332828287755685594822441777128735697886946641091243111
681545238116551858244306426392836846104064731076842543578816818530479152156759
120290446042559868977983946528986507504554880362587909904988045775663409621628
751758337985082711409428127972518695971904771679899146399020169600159539495666
103044643078840967245774105859770442044412377244765493396419787742155086335373

263474448018184154889873746151758484429412090320498815046405341806128814394905
868219934450400886138337839240264255614068324070696593204945639821400860155559
484459548932550840102882387267858402968600171176157376831434777201707285299833
699125037728269831720745214081011576306227271226229939415986451030780497468026
208293099379288210543447284509744576781872462063625033331957011656270150086434
012363118225202497856559102049032543407053333746197469855990322303022144902536
648886278160402357782913621359570037109561666619129528015842754809484320410643
238137797569327685399314897468891322774686848219935886576012631365200360985488
2178324694

Расшифрованный текст:

вот пример статьи на тысячу символов. это достаточно маленький текст, оптимально подходящий для карточек товаров в интернет-магазине 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 картинку. 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 предложениями, союзами и другими частями речи, можно и двести символов, то количество слов не изменно и возрастает. в копирайтерской деятельности принято считать тысячу пробелами или без пробелов. увеличение объёма текста примерно на сто или двести символов имеет настолько разный эффект, что можно разделить слова свободным пространством. считать пробелы заказчики не любят, так как это неустойчивое место. однако некоторые фирмы 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.

Карточка:



Г: ПОТОЧНЫЕ ШИФРЫ

15.A5 /1

A5 — это поточный алгоритм шифрования, используемый для обеспечения конфиденциальности передаваемых данных между телефоном и базовой станцией в европейской системе мобильной цифровой связи GSM (Groupe Spécial Mobile).

Шифр основан на побитовом сложении по модулю два (булева операция «исключающее или») генерируемой псевдослучайной последовательности и шифруемой информации. В A5 псевдослучайная последовательность реализуется на основе трёх линейных регистров сдвига с обратной связью. Регистры имеют длины 19, 22 и 23 бита соответственно. Сдвигами управляет специальная схема, организующая на каждом шаге смещение как минимум двух регистров, что приводит к их неравномерному движению. Последовательность формируется путём операции «исключающее или» над выходными битами регистров.

Код программы:

```
# -*- coding:utf-8 -*-
from base import alphabet, input_for_cipher_short, input_for_cipher_long,
output_from_decrypted
import re
import copy
reg_x_length = 19
reg_y_length = 22
reg_z_length = 23

key_one = ""
reg_x = []
reg_y = []
reg_z = []

def loading_registers(key):
    i = 0
    while(i < reg_x_length):
        reg_x.insert(i, int(key[i]))
        i = i + 1
    j = 0
    p = reg_x_length
    while(j < reg_y_length):
        reg_y.insert(j, int(key[p]))
        p = p + 1
        j = j + 1
    k = reg_y_length + reg_x_length
    r = 0
    while(r < reg_z_length):
        reg_z.insert(r, int(key[k]))
        k = k + 1
        r = r + 1

def set_key(key):
    if(len(key) == 64 and re.match("^[01]+$", key)):
```

```

        key_one = key
        loading_registers(key)
        return True
    return False

def get_key():
    return key_one

def to_binary(plain):
    s = ""
    i = 0
    for i in plain:
        binary = str(' '.join(format(ord(x), 'b') for x in i))
        j = len(binary)
        while(j < 12):
            binary = "0" + binary
            s = s + binary
            j = j + 1
    binary_values = []
    k = 0
    while(k < len(s)):
        binary_values.insert(k, int(s[k]))
        k = k + 1
    return binary_values

def get_majority(x, y, z):
    if(x + y + z > 1):
        return 1
    else:
        return 0

def get_keystream(length):
    reg_x_temp = copy.deepcopy(reg_x)
    reg_y_temp = copy.deepcopy(reg_y)
    reg_z_temp = copy.deepcopy(reg_z)
    keystream = []
    i = 0
    while i < length:
        majority = get_majority(reg_x_temp[8], reg_y_temp[10], reg_z_temp[10])
        if reg_x_temp[8] == majority:
            new = reg_x_temp[13] ^ reg_x_temp[16] ^ reg_x_temp[17] ^
reg_x_temp[18]
            reg_x_temp_two = copy.deepcopy(reg_x_temp)
            j = 1
            while(j < len(reg_x_temp)):
                reg_x_temp[j] = reg_x_temp_two[j-1]
                j = j + 1
            reg_x_temp[0] = new

        if reg_y_temp[10] == majority:
            new_one = reg_y_temp[20] ^ reg_y_temp[21]
            reg_y_temp_two = copy.deepcopy(reg_y_temp)
            k = 1

```

```

        while(k < len(reg_y_temp)):
            reg_y_temp[k] = reg_y_temp_two[k-1]
            k = k + 1
        reg_y_temp[0] = new_one

        if reg_z_temp[10] == majority:
            new_two = reg_z_temp[7] ^ reg_z_temp[20] ^ reg_z_temp[21] ^
reg_z_temp[22]
            reg_z_temp_two = copy.deepcopy(reg_z_temp)
            m = 1
            while(m < len(reg_z_temp)):
                reg_z_temp[m] = reg_z_temp_two[m-1]
                m = m + 1
            reg_z_temp[0] = new_two

        keystream.insert(i, reg_x_temp[18] ^ reg_y_temp[21] ^ reg_z_temp[22])
        i = i + 1
    return keystream

def convert_binary_to_str(binary):
    s = ""
    length = len(binary) - 12
    i = 0
    while(i <= length):
        s = s + chr(int(binary[i:i+12], 2))
        i = i + 12
    return str(s)

def encrypt(plain):
    s = ""
    binary = to_binary(plain)
    keystream = get_keystream(len(binary))
    i = 0
    while(i < len(binary)):
        s = s + str(binary[i] ^ keystream[i])
        i = i + 1
    return s

def decrypt(cipher):
    s = ""
    binary = []
    keystream = get_keystream(len(cipher))
    i = 0
    while(i < len(cipher)):
        binary.insert(i, int(cipher[i]))
        s = s + str(binary[i] ^ keystream[i])
        i = i + 1
    return convert_binary_to_str(str(s))

def user_input_key():
    tha_key = str(input('Введите 64-bit ключ: '))
    if (len(tha_key) == 64 and re.match("^[01]+$", tha_key)):
        return tha_key

```



```

else:
    while(len(tha_key) != 64 and not re.match("^[01]+$", tha_key)):
        if (len(tha_key) == 64 and re.match("^[01]+$", tha_key)):
            return tha_key
        tha_key = str(input('Введите 64-bit ключ: '))
    return tha_key

key = '0101001000011010110001110001100100101001000000110111111010110111'
set_key(key)

print(f'''
A5/1:
Ключ: {key}
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{encrypt(input_for_cipher_short())}

Расшифрованный текст:
{output_from_decrypted(decrypt(encrypt(
    input_for_cipher_short())))}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{encrypt(input_for_cipher_long())}

Расшифрованный текст:
{output_from_decrypted(decrypt(encrypt(
    input_for_cipher_long())))}
''')

```

Тестирование:

```

/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab06_15_a51.py

A5/1:
Ключ: 0101001000011010110001110001100100101001000000110111111010110111
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
110110111010111010100001110101110010000011100101011111001001110110010010101100
000101101110000111110111101000110110110011011100000100010011000101110011111100
110001110110110111110101101011100011111101001110011000001001101010010111101100
100011001101010010111010010100100010001000010101100101001011111011001010111110
001000101001010011111101100001100000010100001001101001100010000101010111100110
0001001101101111000011011100100001000010001000011010111011101100111101011111000

Расшифрованный текст:
время, прилив выиотливы не ждут человека.

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:

```

110110111010111011011111110100000101000011100110011111000110110110011101101100
000110101111110000110110010111110110110010011101111110010011001110110010000110
110000001000110110000110101011100110111101000000011000001001101011100111101101
011010001100101111111010011000100011110110010100010001001011110101001010110110
001001011000010010000001100000011100010100000010101001101011000100101011100111
110001110110110111011010010011001001100011000010101011011100010100101011111100
100110111000000101001101100011111011011110011101111110101101001011101000101010
111010110100011100110000110111011001110010010001111101010000101101101000111100
011011100001111110100111000101101000101001110010101100011011111110001111001100
010110001011000000101101100010011100100001110010010010111011101010100011001110
110111000000010100111001110101010011110000101100000100111101100000100010001111
111111110011010000110000000001101110000000010110111100001011011111000111001001
00100101100011101100010000000010101110011010101011111110111100101100111010011
0000100101111000001101010100100101101110000101010100100010101110100010010000000
001010011010110010110101111011010101101000001101111100011000101100110001
110110111100000010000000011001101000111100110010111110111101101111100011000010
011100000011010111011011100010101100110100100101111001110000111010111010110001
001011011111101011100011110000101000001010000011000100010101001100100000100101
111000111100000000010100000001011001110111000111011010101001100110011100111011
100101000101001010011011001011000110001101110001110110110100000010101000111110
10010011001100110011011010000101011011010111100100100000110000100111011001
001110000100010011011110000110010010000000000101111010101110001000110101111001
011100110001101011000111100011000000010001111010110001100111001111100000010011
101001000101011001100101000010010010111010010101011000100111011000100010111110
11111111011000001010010000011100000101000011000101101101001000111001111111101
011000011000111110000011100101011101101011111110100111011010001001111110000001
010101011100000011011111010110000110111101100110100000100110000011111001010110
101111000001110111011000000100111101001010000101100101011001010001100111111010
011110101011001110111101110101110010101100011100000011010011011100110101000110
111000101000100001001001100010000011100111111011011001011010100001110011011001
101001000000111100000100000001001010110011001001111110001111001101101010000100
001001010010001110100010101001011001010111110001011011100100010110011100100011
101110101101010011001101110111010000111100000011010100110100101101110010101101
000000000101101000100110010111011101010100101110100101010110011101011011000001
101001101101010111010110011111111000001001101011111000101101111010010111010011
111111001111010111100111101001110101010000000011011100111000110010010100001111
110010011011100001100011100010000000100000000110001011010111110110000000001010
000001010111000001010100111001110010100010011011000011111000000011110110100000
101001101100111001000010110110111001101111011000011100110011111100010101100010
101101011000110110001010110110101100111111010100000101111011101111101001101111
01101100010011010011001100100100111010001101010101011101001011000100111111011
010101001111010100111100010001001110110010100011011001011000110111100111100010
1100100100010001101001011111110100111001001110001100011000101011111010111101101
001101001011011100001111110111111011001101000011001001001100101010011101110010
011111101001000100110110011110010100101010000000101101011010111001110110110100
001000000100000011010111010001111111110100101110101111011110000111000000100011
111100010000000110100000101010001101000011110010001000111100011000111001000111
110010010000101001111011100011111000010100010111111010111011000101011001011111
010110101011011001110000001111010000101011110100011001010011100000010011011110
110111100011101001010111011110110000111010101010110001111000000010100100110010
101110001101101110000000000101001111000010010111001010101111111110100011101000
011001101110101011101110001110101011101010110110111110001010101100110001101011
10111110101101101101101001110000000111111101000101010010110001111000000010100100110010

110111000010011001010010110100111011000010010010010011110110010110100100001001
111100101001101111010011011101010101100000100001011100001011001000000001000110
101110000011100011011011000000110001100011001110010001000000100011111001100011
000011011010011110000000001001001011100000101000110110000010001100011001000010
011000011100101100011010110111010001110011011000110011100001001001011010110110
101011001110011111000101100011000110000101101001011100111100010001101001110011
010110111100110001000111110011001000101001100010101101011111100010011001000100
001110000111111010011011111001001001000011110001000011000100100110000001001111
001110111001110011011001010100000010101001110000110010111000001001110111000001
100100100000001010011111011100101000110000110000100011001101010000111111000111
101000100101110110100000001000011110111110011000010110000100100110011100001001
100101001110000111111101011000001110100000100101101101010001000111111001100010
011011100110101011110111010000111110111110111010101100000110001110000001110001
00100110100101111101110011101001010010111010111111000110111111011011011101111
000000100101011010111110111010000110101010100001100101111011001000000111100110
000011111101101111100111000111000001100110110010010101110000000010000111001100
011011011000110101101011011100000001110000000110111010110111000101111001010000
101010010010000000111001001101010101000101011001000010110011111010001110100110
011010110111111100011000001001010011011110100010000010100111010110011111000001
111110001011110010100101011011011100111100000010000101100101000100010001011011
010111010110110111010010110101100001100111101011010000101000000000100101100111
100101101011100001011111010011011000000101001111110111001000100000001111000001
011110100111011101000101101001101001110101010011110111100000100001100100111111
101101010110010101001110001110111000010011011000110011000011101000001000110111
110010010101001000111111111011001011110110110000011011100111011101001111000110
110111011100101011010101100110011010000010010111001010101101110101011101101101
111001001011110100010101011110011001010010010010111000111010110000011111111111001
111100100000100001111101000110000010111000111001001010010010010101111101110111
011001011010101101100101100001001011111101000010001111100000110110110000010011
111100001010001010000101010100011110111000001110000100101101010111100010111000
010010010000011000001111011111100011111110011011101010011111111010101101110101
100110011100110100010011110111100010010000010010011110000101011100101011011100
111000000110010010100011000010101101110011101011111010000000010010001111011101
001000110110111001111001000011011101001001100110100000111010011010110000000111
011111100011011010010010011000010000110101001110000101110100011011000000100110
0010001110110011011000101001111111000000000010110111011010011101110001010100010
000111100011010100000100101110110001000110010101111100101101000011010010011000
111011000010001100001001101100000100000110101001100000110010010011111011011100
111011001001000111111011001011101101011100100100111110010001001101110110100100
011110000100011101100111000000011011000011110001110011010000001011010100000110
011110100101110100001110111111101000101010110011101111101100111000110001001101
0010010000000000001110010011111010110000011010010010100110011111000000101010001
101100111101100010111010111011000110001001011000110011110100010011010001101101
011001000101111101000101001010100011011111011011000011010010000011111110011000
100100101000000100101000011100111100001110110011111001000111100110000001010001
100001011011100110011101100110100000000100101001111011101111110000100100011000
011011110001000011110110100010100101101010000101001001110001011001111000101001
100010000010100001011000101100101100101011000001110100001101001100011011110100
000000111011100001010001101010110101001111100000101110010011110110000100010011
110001100010010100111100101100001011011000011001010011110110100111000000111100
100111001001010100000111101010010100111010101001100111111100001000110101101100
000110100010100110011011111011010101001110101011000001000011111000001001110000
101010010000000111000101101110011110100001001001111011111010101110001001010111

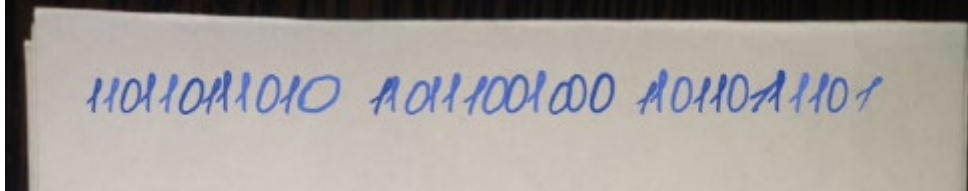
111001110001100110010110100111000001110010000111011001100101111001000000111011
110011000011001010101000000010110010100001011010101111010101010001111111011101
101001000010011110111100011100011100111010100110101110011101101101111111001110
011010100001010101110100011110111011100000110011010110000001110100011001100110
111010100010101011001010110101110000000010001001110010001111010001111000010000
100010000111100101011010001100110010001010010000010101001011101111111111011011
110110001101010010011110101010101001011100011011110100110001110101100000110111
000000100111101011111101001000011000110011111000001010110110011001010010001010
101010111011110110100101000010111010101110101101001000001000000110101100000101
101100011000000001001011101101001001111111001011011001100010011101011010000001
111000101100000101010000111100100111101101001000000000111000001100001101001000
011011111010101100111110111010111111111011001111100010111010000010111111110000
011000001001111110101000001100110111101011101101010011000111100111111011100001
0110000110010011000000000001111011111001010010101011001110010010000110011001011
111001110011011111000000011000001000001001000010001111101011100011100100110101
101110001000100101100100110000000001000111111100001110011001111111111010010110
110010100000010101000100000101010001000010011100100110110111000010011010110110
110100001000011110111000110001000011101000101100000000011001001111111101001111
100100001110001010000010110101111010100010000111111010011010101110100001001010
100110010100000001100011100001110011001101011010000000010100010100010000101111
011011101101000110100101110010010110111011010010001110110001111100100010011111
110000000110000011000010111001000100111001000100000011011101011110101101100110
100110111011110000111110010001100000100001010101011100111001111010111110100101
101000110111011111111001000110001000100111001100111010111100011001101001101101
101101101001001101001111011111100000101001001110000100110101110011010001111110
110000011110101010101000011101011111100110000000001010101011010110011100100111
001001101100010101110101110100000010010001100100111000111011100001111000100111
111001110000001000010000111001110001010001010000111001110110111000000100110101
100111001110011110101011101001101101000001010101000011100010010010100000011011
0111111010101101100101100001010011110000110011000111111010000100100111000111101
100110101010010100000111000010100101001110101010000011011100110000010110100010
011000101000110001111110100101111010000100100010001110010110101000111011111111
0000110111001001111111110101010111100000010110000111011100001010011010101000000
010011001101101101010001101100010111010001111001000000100010000110101110111111
100100100011010001100111110110101110011000001110110111011110100010110110011111
000110111100100111011011100000001000011000010000011001001111010001111001001011
011111111001011010010101111100010000101010100010000001010000101101110111111111
1011011100110000000001011100000111100111111111111110101110111010010001111000110
010101010001101110001011011111001010011001101110011001111000110011001011001000
100001011011011111100000000100011011110000110001001111111000001000100101110010
1000000000001010100000010111101100101111011010001010110011100100010010000011111
100100010001111011001001000011110100001000001011001010111101100011011110010100
0111001001111100001100100011101100000011011010111111110101111110101111110011001
0110111011001110111001101111000010000010101010000110100000111101111001011000100
011111001000111001000000110011101101010100111010100110001001101001010000010111
111000111000010100010101010111010010110000010100110010011110111101010100011010
010110010110110010100110010100000000010111001110101010111101100110101001001000
010101100010010111010110110100110100000011010100000111100100000100001001011001
101100011100010011011001011111011000000101111111111100010010110011001001111110
1011110100001100100100110001110100110011001111111110010101111110110000011000100
001000100110100011110010101101010111000001001010001110000111001000011110111000
111010110011110100001010001010101100010010001111011110010001011001001011001101
1000011000000100011100010110011001100001111011100100010110010010110011011111

100010010100000110110100111100011000111001011101010010101110001010110011111111
101000110111000011101010010110001101110011110111000001001001001101010011111101
001000111100010000010110111110111001000101000001100011110100101010100111001001
0001011111000100101100011001011101111001110100101010000001000000100010101101011
011101001011000000111111010111000100101110001110110000101011001000010101100110
000001101111001100111100000100110011111110011011100100011011110011011010110010
011101010001101100111000101111010100000111001000100101101111000110001001101101
000100110001000001101001110111010111011010111011101001001100110000110010011111
10101010101011011010

Расшифрованный текст:

вот пример статьи на тысячу символов. это достаточно маленький текст, оптимально подходящий для карточек товаров в интернет-магазине 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 картинку. текст на тысячу символов это сколько примерно слов. статистика показывает, что тысяча включает в себя столько слов, сколько требуется для среднего размера статьи. но, если злоупотреблять предложениями, союзами и другими частями речи, можно и один или два символа, то количество слов не изменно и возрастает. в копиях итерской деятельности принято считать тысячу пробелами или без. учет пробелов увеличивает объём текста примерно на столько, сколько и два символа, то количество слов не изменно и возрастает. разделяем слова свободным пространством. считать пробелы заказчики не любят, так как это неустойчивое место. однако некоторые фирмы 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.

Карточка:



16.A5 /2

A5 — это поточный алгоритм шифрования, используемый для обеспечения конфиденциальности передаваемых данных между телефоном и базовой станцией в европейской системе мобильной цифровой связи GSM (Groupe Spécial Mobile).

Шифр основан на побитовом сложении по модулю два (булева операция «исключающее или») генерируемой псевдослучайной последовательности и шифруемой информации. В A5 псевдослучайная последовательность реализуется на основе трёх линейных регистров сдвига с обратной связью. Регистры имеют длины 19, 22 и 23 бита соответственно. Сдвигами управляет специальная схема, организующая на каждом шаге смещение как минимум двух регистров, что приводит к их неравномерному движению. Последовательность формируется путём операции «исключающее или» над выходными битами регистров.

Код программы:

```
# -*- coding:utf-8 -*-
from base import alphabet, input_for_cipher_short, input_for_cipher_long,
output_from_decrypted
import sys
import copy
import re

reg_x_length = 19
reg_y_length = 22
reg_z_length = 23
reg_e_length = 17

key_one = ""
reg_x = []
reg_y = []
reg_z = []
reg_e = []

def loading_registers(key):
    i = 0
    while(i < reg_x_length):
        reg_x.insert(i, int(key[i]))
        i = i + 1
    j = 0
    p = reg_x_length
    while(j < reg_y_length):
        reg_y.insert(j, int(key[p]))
        p = p + 1
        j = j + 1
    k = reg_y_length + reg_x_length
    r = 0
    while(r < reg_z_length):
        reg_z.insert(r, int(key[k]))
        k = k + 1
        r = r + 1
    i = 0
```

```

while(i < reg_e_length):
    reg_e.insert(i, int(key[i]))
    i = i + 1

def set_key(key):
    if(len(key) == 64 and re.match("^[01]+$", key)):
        key_one = key
        loading_registers(key)
        return True
    return False

def get_key():
    return key_one

def to_binary(plain):
    s = ""
    i = 0
    for i in plain:
        binary = str(' ').join(format(ord(x), 'b') for x in i)
        j = len(binary)
        while(j < 12):
            binary = "0" + binary
            s = s + binary
            j = j + 1
    binary_values = []
    k = 0
    while(k < len(s)):
        binary_values.insert(k, int(s[k]))
        k = k + 1
    return binary_values

def get_majority(x, y, z):
    if(x + y + z > 1):
        return 1
    else:
        return 0

def get_keystream(length):
    reg_x_temp = copy.deepcopy(reg_x)
    reg_y_temp = copy.deepcopy(reg_y)
    reg_z_temp = copy.deepcopy(reg_z)
    reg_e_temp = copy.deepcopy(reg_e)
    keystream = []
    i = 0
    while i < length:
        majority = get_majority(reg_e_temp[3], reg_e_temp[7], reg_e_temp[10])
        if get_majority(reg_x_temp[12], reg_x_temp[14], reg_x_temp[15]) ==
majority:
            new = reg_x_temp[13] ^ reg_x_temp[16] ^ reg_x_temp[17] ^
reg_x_temp[18]
            reg_x_temp_two = copy.deepcopy(reg_x_temp)
            j = 1
            while(j < len(reg_x_temp)):

```

```

        reg_x_temp[j] = reg_x_temp_two[j-1]
        j = j + 1
    reg_x_temp[0] = new

    if get_majority(reg_y_temp[9], reg_y_temp[13], reg_y_temp[16]) ==
majority:
        new_one = reg_y_temp[20] ^ reg_y_temp[21]
        reg_y_temp_two = copy.deepcopy(reg_y_temp)
        k = 1
        while(k < len(reg_y_temp)):
            reg_y_temp[k] = reg_y_temp_two[k-1]
            k = k + 1
        reg_y_temp[0] = new_one

    if get_majority(reg_z_temp[13], reg_z_temp[16], reg_z_temp[18]) ==
majority:
        new_two = reg_z_temp[7] ^ reg_z_temp[20] ^ reg_z_temp[21] ^
reg_z_temp[22]
        reg_z_temp_two = copy.deepcopy(reg_z_temp)
        m = 1
        while(m < len(reg_z_temp)):
            reg_z_temp[m] = reg_z_temp_two[m-1]
            m = m + 1
        reg_z_temp[0] = new_two

    keystream.insert(i, reg_x_temp[18] ^ reg_y_temp[21] ^ reg_z_temp[22])
    i = i + 1
return keystream

def convert_binary_to_str(binary):
    s = ""
    length = len(binary) - 12
    i = 0
    while(i <= length):
        s = s + chr(int(binary[i:i+12], 2))
        i = i + 12
    return str(s)

def encrypt(plain):
    s = ""
    binary = to_binary(plain)
    keystream = get_keystream(len(binary))
    i = 0
    while(i < len(binary)):
        s = s + str(binary[i] ^ keystream[i])
        i = i + 1
    return s

def decrypt(cipher):
    s = ""
    binary = []
    keystream = get_keystream(len(cipher))
    i = 0

```



```

while(i < len(cipher)):
    binary.insert(i, int(cipher[i]))
    s = s + str(binary[i] ^ keystream[i])
    i = i + 1
return convert_binary_to_str(str(s))

key = '0101001000011010110001110001100100101001000000110111111010110111'
set_key(key)

print(f'''
A5/2:
Ключ: {key}
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{encrypt(input_for_cipher_short())}

Расшифрованный текст:
{output_from_decrypted(decrypt(encrypt(
    input_for_cipher_short())))}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{encrypt(input_for_cipher_long())}

Расшифрованный текст:
{output_from_decrypted(decrypt(encrypt(
    input_for_cipher_long())))}
''')

```

Тестирование:

```

/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab06_16_a52.py

A5/2:
Ключ: 0101001000011010110001110001100100101001000000110111111010110111
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
010000110010010001000000010000110101010000111100010001001111010000110111010000
111111010001000010010000111111010001000000010000111000010000111011010000111000
010000110010010001001011010000111000010000111110010001000010010000111011010000
111000010000110010010001001011010000111101010000110101010000110110010000110100
010001000011010001000010010001000111010000110101010000111011010000111110010000
110010010000110101010000111010010000110000010001000010010001000111010000111010

Расшифрованный текст:
время, прилив и отлив вынежут человека.

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:

```

010000110010010000111110010001000010010000111111010001000000010000111000010000
111100010000110101010001000000010001000001010001000010010000110000010001000010
010001001100010000111000010000111101010000110000010001000010010001001011010001
000001010001001111010001000111010001000011010001000001010000111000010000111100
010000110010010000111110010000111011010000111110010000110010010001000010010001
000111010000111010010001001101010001000010010000111110010000110100010000111110
010001000001010001000010010000110000010001000010010000111110010001000111010000
111101010000111110010000111100010000110000010000111011010000110101010000111101
010001001100010000111010010000111000010000111001010001000010010000110101010000
11101001000100000101000100001001000011011101000011111010001000010010000111110
010000111111010001000010010000111000010000111100010000110000010000111011010001
001100010000111101010000111110010000111111010000111110010000110100010001000101
010000111110010000110100010001001111010001001001010000111000010000111001010000
110100010000111011010001001111010000111010010000110000010001000000010001000010
010000111110010001000111010000110101010000111010010001000010010000111110010000
110010010000110000010001000000010000111110010000110010010000110010010000111000
010000111101010001000010010000110101010001000000010000111101010000110101010001
000010010000111000010000111011010000111000010000111100010000110000010000110011
010000110000010000110111010000111000010000111101010000110000010001000101010000
111000010000111011010000111000010000110100010000111011010001001111010000111101
010000110101010000110001010000111110010000111011010001001100010001001000010000
111000010001000101010000111000010000111101010001000100010000111110010001000000
0100001111100010000110000010001000110010000111000010000111110010000111101010000
111101010001001011010001000101010000111111010001000011010000110001010000111011
0100001110000100001110100100001100000100010001110010000111000010000111001010001
000010010001000111010000111010010000110010010001000010010000110000010000111010
010000111110010000111100010001000010010000110101010000111010010001000001010001
000010010000110101010001000000010000110101010000110100010000111010010000111110
010000110001010001001011010000110010010000110000010000110101010001000010010000
110001010000111110010000111011010000110101010000110101010000110100010000110010
010001000011010001000101010000111000010000111011010000111000010001000010010001
000000010001010001010001000101010000110000010000110001010000110111010000110000
010001000110010000110101010000110010010000111000010000111110010000110001010001
001011010001000111010000111101010000111110010000111110010000110100010000111000
0100001111101010000111111010000111110010000110100010000110111010000110000010000
110011010000111110010000111011010000111110010000110010010000111110010000111010
010001000010010001000111010000111010010000111101010000111110010000111100010000
111110010000110110010000111101010000111110010000111000010000110001010000110101
010000110111010000111101010000110101010000110011010000111110010001000010010001
000111010000111010010000111101010000110000010001000010010001001011010001000001
010001001111010001000111010001000011010001000001010000111000010000111100010000
110010010000111110010000111011010000111110010000110010010001000000010000110101
010000111010010000111110010000111100010000110101010000111101010000110100010000
111110010000110010010000110000010000111101010000111110010000111000010001000001
010000111111010000111110010000111011010001001100010000110111010000111110010000
110010010000110000010001000010010001001100010000111110010000110100010000111000
010000111101010000111000010000111011010000111000010000110100010000110010010000
110000010000111010010000111011010001001110010001000111010000110000010000111000
010000111110010000110100010000111101010001000011010000111010010000110000010001
000000010001000010010000111000010000111101010001000011010001000010010001000111
010000111010010001000010010000110101010000111010010001000001010001000010010000
111101010000110000010001000010010001001011010001000001010001001111010001000111
010001000011010001000001010000111000010000111100010000110010010000111110010000

111011010000111110010000110010010001001101010001000010010000111110010001000001
010000111010010000111110010000111011010001001100010000111010010000111110010000
111111010001000000010000111000010000111100010000110101010001000000010000111101
010000111110010001000001010000111011010000111110010000110010010001000010010001
000111010000111010010001000001010001000010010000110000010001000010010000111000
010001000001010001000010010000111000010000111010010000110000010000111111010000
111110010000111010010000110000010000110111010001001011010000110010010000110000
010000110101010001000010010000110111010000111111010001000010010001000111010001
000010010000111110010001000010010001001011010001000001010001001111010001000111
010000110000010000110010010000111010010000111011010001001110010001000111010000
110000010000110101010001000010010000110010010001000001010000110101010000110001
010001001111010001000001010001000010010000111110010000111111010001001111010001
000010010001001100010000110100010000110101010001000001010001001111010001000010
01000011110000100001110111010000111000010000110100010000110010010000110101010001
000001010001000010010000111000010001000001010000111011010000111110010000110010
010001000001010001000000010000110101010000110100010000111101010000110101010000
111001010000110010010000110101010000111011010000111000010001000111010000111000
010000111101010001001011010001000010010001000111010000111010010000111101010000
111110010000110111010000111111010001000010010000110101010001000001010000111011
010000111000010000110111010000111011010000111110010001000011010000111111010000
111110010001000010010001000000010000110101010000110001010000111011010001001111
010001000010010001001100010000111111010001000000010000110101010000110100010000
111011010000111110010000110011010000110000010000111100010000111000010000110111
010000111111010001000010010001000001010000111110010001001110010000110111010000
110000010000111100010000111000010000111000010000110100010001000000010001000011
010000110011010000111000010000111100010000111000010001000111010000110000010001
000001010001000010010001001111010000111100010000111000010001000000010000110101
010001000111010000111000010000111101010000110000010000111110010000110100010000
111000010000111101010000111000010000111011010000111000010000110100010000110010
010000110000010001000001010000111000010000111100010000110010010000111110010000
111011010000110000010000110111010000111111010001000010010001000010010000111110
010000111010010000111110010000111011010000111000010001000111010000110101010001
000001010001000010010000110010010000111110010001000001010000111011010000111110
010000110010010000111101010000110101010000111000010000110111010000111100010000
110101010000111101010000111101010000111110010000110010010000111110010000110111
010001000000010000110000010001000001010001000010010000110000010000110101010001
000010010001000010010001000111010000111010010000110010010000111010010000111110
010000111111010000111000010001000000010000110000010000111001010001000010010000
110101010001000000010001000001010000111010010000111110010000111001010000110100
010000110101010001001111010001000010010000110101010000111011010001001100010000
111101010000111110010001000001010001000010010000111000010000111111010001000000
010000111000010000111101010001001111010001000010010000111110010001000001010001
000111010000111000010001000010010000110000010001000010010001001100010001000010
010001001011010001000001010001001111010001000111010000111000010001000001010000
111111010001000000010000111110010000110001010000110101010000111011010000110000
0100001111100010000111000010000111000010000111011010000111000010000110001010000
110101010000110111010001000010010001000111010000111010010001000011010001000111
010000110101010001000010010000111111010001000000010000111110010000110001010000
110101010000111011010000111110010000110010010001000011010000110010010000110101
010000111011010000111000010001000111010000111000010000110010010000110000010000
110101010001000010010000111110010000110001010001001010010000110101010000111100
010001000010010000110101010000111010010001000001010001000010010000110000010000
111111010001000000010000111000010000111100010000110101010001000000010000111101

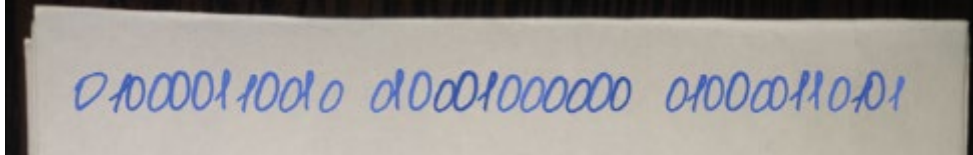
010000111110010000111101010000110000010001000001010001000010010000111110010000
111000010000111011010000111000010000110100010000110010010000110101010001000001
010001000010010000111000010001000001010000111000010000111100010000110010010000
111110010000111011010000111110010000110010010000111000010000111100010000110101
010000111101010000111101010000111110010001000001010001000010010000111110010000
111011010001001100010000111010010000111110010001000000010000110000010000110111
010000111100010001001011010001000000010000110000010000110111010000110100010000
110101010000111011010001001111010000110101010000111100010001000001010000111011
010000111110010000110010010000110000010001000001010000110010010000111110010000
110001010000111110010000110100010000111101010001001011010000111100010000111111
010001000000010000111110010001000001010001000010010001000000010000110000010000
111101010001000001010001000010010000110010010000111110010000111100010001000010
010001000111010000111010010001000001010001000111010000111000010001000010010000
110000010001000010010001001100010000111111010001000000010000111110010000110001
010000110101010000111011010001001011010000110111010000110000010000111010010000
110000010000110111010001000111010000111000010000111010010000111000010000111101
010000110101010000111011010001001110010000110001010001001111010001000010010000
110111010000111111010001000010010001000010010000110000010000111010010000111010
010000110000010000111010010001001101010001000010010000111110010000111111010001
000011010001000001010001000010010000111110010000110101010000111100010000110101
010001000001010001000010010000111110010001000010010001000111010000111010010000
111110010000110100010000111101010000110000010000111010010000111110010000111101
010000110101010000111010010000111110010001000010010000111110010001000000010001
001011010000110101010001000100010000111000010001000000010000111100010001001011
010000111000010000110001010000111000010001000000010000110110010000111000010000
110010010000111000010000110100010001001111010001000010010001000001010000111111
010001000000010000110000010000110010010000110101010000110100010000111011010000
111000010000110010010001001011010000111100010001000001010001000010010000110000
010000110010010000111000010001000010010001001100010001000001010001000010010000
111110010000111000010000111100010000111110010001000001010001000010010001001100
010000110111010000110000010001000010010001001011010001000001010001001111010001
000111010001000011010001000001010000111000010000111100010000110010010000111110
010000111011010000111110010000110010010001000001010000111111010001000000010000
111110010000110001010000110101010000111011010000110000010000111100010000111000
010000110111010000111111010001000010010001000001010001000111010000111000010001
000010010000110000010001001111010000111111010000111110010001000001010000111011
010000110101010000110100010000111101010000111000010000110101010000110010010000
110000010000110110010000111101010001001011010000111100010001001101010000111011
010000110101010000111100010000110101010000111101010001000010010000111110010000
111100010000111010010000110000010001000111010000110101010001000001010001000010
010000110010010000110101010000111101010000111101010000111110010000110011010000
111110010000110010010000111110010001000001010000111111010001000000010000111000
010001001111010001000010010000111000010001001111010001000010010001000111010000
111010010001000001010000111110010000110011010000111011010000110000010001000001
010000111000010001000010010000110101010001000001010001001100010000110111010000
111111010001000010010001000111010000111000010001000010010000110000010001000010
010001001100010001000001010000111011010000111000010001000010010000111101010001
001011010000111001010001000010010000110101010000111010010001000001010001000010
010000110001010000110101010000110111010000110101010000110100010000111000010000
111101010000111110010000110011010000111110010000111111010001000000010000111110
010000111111010001000011010001000001010000111010010000110000010000110111010000
111111010001000010010000111101010000111000010000111010010001000010010000111110
01000011110101000011010001000001010000111010010000110000010000110111010000
111111010001000010010000111101010000111000010000111010010001000010010000111110
010000111101010000110101010000110001010001000011010000110100010000110101010001

000010010001000010010001000111010000111010010000111101010000111110010000110001
010000111110010000111011010001001100010001001000010000111000010000111101010001
000001010001000010010000110010010001000011010000111101010001000011010000110110
010000111101010000110000010001000110010000110101010000111101010000110000010000
110111010000110000010001000010010001001011010001000001010001001111010001000111
010001000011010000110111010000111101010000110000010000111010010000111110010000
11001001000011000101000011010101000011011101000011111010001000000010000111110
010000110001010000110101010000111011010000111110010000110010010001000010010001
000111010000111010

Расшифрованный текст:

вот пример статьи на тысячу символов. это достаточно маленький текст, оптимально подходящий для карточек товаров в интернет или магазина 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 картинку. текст на тысячу символов это сколько примерно слов. статистика показывает, что тысяча включает в себя пятьдесят или двести слов средней величины. но, если злоупотреблять предложениями, союзами и другими частями речи на один или два символа, то количество слов не изменно и возрастает. в копирайтерской деятельности принято считать тысячу пробелами или без. учет пробелов увеличивает объём текста примерно на сто или двести символов именно столько раз мы разделяем слова свободным пространством. считать пробелы заказчики не любят, так как это неустойчивое место. однако некоторые фирмы биржи видят справедливым ставить стоимость за тысячу символов 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.

Карточка:



Блок G: КОМБИНАЦИОННЫЕ ШИФРЫ

17.МАГМА

Магма представляет собой симметричный блочный алгоритм шифрования с размером блока входных данных 64 бита, секретным ключом 256 бит и 32 раундами шифрования.

Код программы:

```
from base import alphabet, input_for_cipher_short, input_for_cipher_long,
output_from_decrypted

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

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

def rot11(x):
    return ((x << 11) ^ (x >> (32 - 11))) & MASK32

def g(x, k):
    return rot11(t((x + k) % 2 ** 32))

def split(x):
    L = x >> 32
    R = x & MASK32
    return (L, R)

def join(L, R):
    return (L << 32) ^ R

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

def magma_encrypt(x, k):
    keys = magma_key_schedule(k)
    (L, R) = split(x)
    for i in range(31):
        (L, R) = (R, L ^ g(R, keys[i]))
    return join(L ^ g(R, keys[-1]), R)

def magma_decrypt(x, k):
    keys = magma_key_schedule(k)
    keys.reverse()
    (L, R) = split(x)
    for i in range(31):
        (L, R) = (R, L ^ g(R, keys[i]))
    return join(L ^ g(R, keys[-1]), R)

key = int('ffeeddcbbaa99887766554433221100f0f1f2f3f4f5f6f7f8f9fafbfcfdfeff',
16)

i = 0
text_short = input_for_cipher_short()
encr_short = []
while (i < len(text_short)):
    text = text_short[i:i+4].encode().hex()
    text = int(text, 16)
    text = text % 2**64
    pt = text
    ct = magma_encrypt(pt, key)
    encr_short.append(ct)
    i += 4
decr_short = []
for i in encr_short:
    dt = magma_decrypt(i, key)
    decr_short.append(bytes.fromhex(hex(dt)[2::]).decode('utf-8'))

i = 0
text_long = input_for_cipher_long()
encr_long = []
while (i < len(text_long)):
    text = text_long[i:i+4].encode().hex()
    text = int(text, 16)
    text = text % 2**64
    pt = text
    ct = magma_encrypt(pt, key)
    encr_long.append(ct)
    i += 4
decr_long = []
for i in encr_long:

```

```

dt = magma_decrypt(i, key)
decr_long.append(bytes.fromhex(hex(dt) [2::]) .decode('utf-8'))

print(f'''
МАГМА:
КЛЮЧ:
{key}

КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{encr_short}

Расшифрованный текст:
{output_from_decrypted(''.join(decr_short))}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{encr_long}

Расшифрованный текст:
{output_from_decrypted(''.join(decr_long))}
''')

```

Тестирование:

```

/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab07_17_magma.py

МАГМА:
КЛЮЧ:
115761816795685524522806652725025505786200410505847444308688553892001406123775

КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
[18432907413224455314, 10996816857283808610, 1603220777717569738,
6798339374272425273, 625275379878570582, 12897841916972840738,
12693135464871535956, 6338906346771095526, 2952080121925535959,
10585345143535530769]

Расшифрованный текст:
время, приливьиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
[8343875602038808058, 8041676007725686027, 7453366501928099122,
13622418350652450349, 13257185717927971463, 3967957328028735447,
6303045106133971510, 17694607534926283087, 3147373830694159915,
9390122889656481141, 15485378634152172683, 11157654134498461325,
9103621146938759596, 18167786785284461467, 4720588269896140616,
13198230271257633374, 290271297514756748, 6383750138269828222,
6966411025704842352, 1184081237020962173, 5518298843322725716,
1121075660661397656, 10048741617664022091, 18363685243888377546,
95605295917991907, 17057631210449679753, 16551488750349984268,

```


2025585132759738150, 12949061887509852732, 9329546944062831470,
3418244764798143790, 15295462055836379806, 5155680359283034501,
9451937380501232912, 7328220556957620599, 3614937074236676557,
4589181499513045879, 13803166415433109120, 18278897981036660435,
4019936986318215394, 12394731635219602309, 5416248858033923816,
463232828950025933, 15989759834534178049, 13360939273674915028,
8528608555291906000, 10293453731533650333, 13238975415012678998,
1261680965272169368, 10483829524585249195, 15703560336104773581,
2928612788789974786, 12370719347969566168, 15498018845651124401,
7975202005615435922, 1001640210394968931, 9482390876876309460,
10306886503249707528, 3432371120083104576, 13284723522159984134,
4732957936308484548, 18145150074698797421, 7665912658016440955,
17186319045640305491, 13257185717927971463, 3967957328028735447,
6303045106133971510, 1163720297091009342, 2346699712756694809,
10356924971308547229, 13990735642799502546, 16656837303721777190,
3620970884711728977, 9443487292630364151, 7691049094682026909,
916112073177174944, 15797466941996902229, 16777509928585489639,
6303121470213798711, 11674884899395813190, 18046574429007017128,
13077766683806041775, 6569214857954439428, 13257185717927971463,
3967957328028735447, 6303045106133971510, 18131068278186858513,
16002418093945299027, 2908114868778074520, 11794392602089101467,
1199691821864277191, 3076938350476950006, 13450817471486594379,
10728741519359495825, 9269189682480091597, 14111778268298096419,
2057896382844501704, 10293453731533650333, 17204212421686848739,
5366206226868800702, 8248466997448536722, 10088810565957275850,
2254355090243742257, 8371896884399298133, 10985518332543285851,
17941711041904849701, 7740200207943279354, 10750419551538638209,
2629297776519959089, 2225742027093608502, 287244974227193751,
17712479779128577940, 9222296997056994615, 973788500067181560,
3393954245381334839, 10641483480207737426, 16269104980183698483,
16250235473790191753, 2700688729269881630, 8411607884348754050,
7636448048795956798, 692746442652315515, 11343070527907873916,
629521920599350430, 14623128418242633989, 5705247633908105959,
17910568456840661856, 16201319981229519019, 13769298882463179721,
7421154077805485456, 9312183877157555560, 7691049094682026909,
916112073177174944, 15135398776342705892, 15289260373132081671,
7126379060141568292, 1725036945967272673, 17688755856354297123,
138113508643393643, 14127475102588350875, 4151016356341159175,
9841467487393741636, 3187959327495924697, 15728081324422447459,
16055605958376054530, 3572266014017346359, 3223827000587680474,
8253733500166311251, 9140508542287809951, 1790032636239692933,
14268543881935307321, 7342482303792008383, 17677570963908781400,
13355264069952555938, 88597409000583480, 17257030184527390406,
12715322944275189718, 4311305660788670662, 7008462056628757749,
5705247633908105959, 780471545777017295, 7181337429584918154,
17602858510462705198, 14738061815484779755, 540299310016161264,
10182683375894303875, 8026343204551337982, 17660378940581534682,
13825198160625090606, 4720588269896140616, 5730527409401222056,
17941293450735671408, 4342366351984147086, 5313276448158165379,
1745024168316228346, 2528136519964988665, 3547755811857389063,
10306886503249707528, 4526494865159228242, 7342482303792008383,
2908114868778074520, 7471680038596777270, 16412799349039310306,
17456146687817481297, 2894321475946271648, 6923158161737971479,

10027397076122517884, 11785376343832896276, 4548889844418068535,
9068186136604959843, 17006372751687582512, 9582236842043709584,
2756346197016409968, 12319862187853716832, 14738061815484779755,
3597986015127692257, 6006230909502237940, 8049089039016717789,
6157047194034613233, 6958374052664244197, 16431478885293681293,
8935011217054641103, 4493811008412807767, 2950821783567146896,
1855179961465792476, 6714524409280649005, 13570818025122198173,
14348383310813638904, 4453067278648320496, 3438244984932255310,
9108046240042143694, 151270379504886441, 10183946694284410805,
580256645846205130, 5356452764985505955, 1949584005305844760,
5054509616596506897, 806561947066774048, 14232614625982377749,
4486402459089481420, 6142067374720541917, 11774712973292423210,
10305629430321463988, 3547755811857389063, 10306886503249707528,
15489555906471766183, 15441006425873291865, 2659805083408552087,
12582666581260422456, 541882649396588128, 4886609086442669621,
5561196856767141965, 7731325888395147745, 1096626906536588642,
763947604362793674, 12798794780496052712, 8299242569789435168,
6766061707386831762, 8971528550958227067, 11254181301937016096,
4106793478691170180, 5812305374465111022, 14122431208396972645,
8756880839685289505, 1840145209499476946, 17204212421686848739,
12702832949266190125, 12312436585117905032, 2731161190011158263,
4720588269896140616, 13119149454917978330, 9822108851446653348,
4085677428333094007, 12601379616160262740, 10265087333462641478,
8770956180316534333, 2746212430076756819, 10780803050552638165,
1906124277192103447, 8318790287988703211, 9451937380501232912,
7154010589201716226, 6593238725915939856, 16381960257093506421,
16805496448556233763, 13257185717927971463, 3534238871897283987,
18436104050552264826, 3237166291520387028, 14738061815484779755,
540299310016161264, 10585345143535530769]

Расшифрованный текст:

вот пример статьи на тысячу символов. это достаточно маленький текст, оптимально подходящий для карточек товаров в интернет или магазина 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 картинку. текст на тысячу символов это сколько примерно слов. статистика показывает, что тысяча включает в себя столько десяти или двести слов средней величины. но, если злоупотреблять предложениями, союзами и другими частями речи на один или два символа, то количество слов не изменно и возрастает. в копирайтерской деятельности принято считать тысячу пробелами или без учета пробелов и 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 неустойчивое место. однако некоторые фирмы 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.

Карточка:

1 8 4 3 2 9 0 7 4 1 3
2 2 4 4 5 5 3 1 4

БЛОК Н: АСИММЕТРИЧНЫЕ ШИФРЫ

21.RSA

Берутся два очень больших простых числа P и Q и находится произведение простых чисел $N=P \times Q$ и функция Эйлера от этого произведения $\phi(N)=(P-1) \times (Q-1)$. Выбирается случайное целое число E , взаимно простое с $\phi(N)$, и вычисляется $D=(1 \bmod \phi(N))/E$. E и N публикуются как открытый ключ, D сохраняется в тайне. Шифрование: Если M — сообщение, то шифртекст C_i получается последовательным шифрованием каждой шифрвеличины M_i возведением ее в степень E по модулю N : $C_i = M_i^E \bmod N$. Расшифрование: Получатель расшифровывает сообщение, возводя последовательно C_i в степень D по модулю N : $M_i = C_i^D \bmod N$.

Код программы:

```
# -*- coding:utf-8 -*-
from base import alphabet, input_for_cipher_short, input_for_cipher_long,
output_from_decrypted
import random

def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

def multiplicative_inverse(e, r):
    for i in range(r):
        if (e*i) % r == 1:
            return i

def is_prime(num):
    if num == 2:
        return True
    if num < 2 or num % 2 == 0:
        return False
    for n in range(3, int(num**0.5)+2, 2):
        if num % n == 0:
            return False
    return True

def generate_keypair(p, q):
    if not (is_prime(p) and is_prime(q)):
        raise ValueError('Оба числа должны быть простыми.')
    elif p == q:
        raise ValueError('p и q не могут быть равны друг другу')
    n = p * q

    phi = (p-1) * (q-1)

    e = random.randrange(1, phi)

    g = gcd(e, phi)
```

```

while g != 1:
    e = random.randrange(1, phi)
    g = gcd(e, phi)
d = multiplicative_inverse(e, phi)
return ((e, n), (d, n))

def encrypt(pk, plaintext):
    key, n = pk
    cipher = [(ord(char) ** key) % n for char in plaintext]
    return cipher

def decrypt(pk, ciphertext):
    key, n = pk
    plain = [chr((char ** key) % n) for char in ciphertext]
    return ''.join(plain)

p = 107
q = 109

public, private = generate_keypair(p, q)

message_short = input_for_cipher_short()
encrypted_short = encrypt(private, message_short)
print_enc_short = ''.join([str(x) for x in encrypted_short])
decrypted_short = decrypt(public, encrypted_short)

message_long = input_for_cipher_long()
encrypted_long = encrypt(private, message_long)
print_enc_long = ''.join([str(x) for x in encrypted_long])
decrypted_long = decrypt(public, encrypted_long)

print(f'''
RSA:
Ключ:
p={p} q={q}
Публичный: {public}
Приватный: {private}
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{print_enc_short}

Расшифрованный текст:
{output_from_decrypted(decrypted_short)}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{print_enc_long}

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

Тестирование:

```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab08_21_rsa.py
```

```
_
```

RSA:

Ключ:

p=107 q=109

Публичный: (5441, 11663)

Приватный: (4025, 11663)

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

759271718847938599013012565910028565971711881804188759210654188633810028180418
875921065484778847319610754100291002864568847180463387592884725481001810028645
62548

Расшифрованный текст:

время, приливьиотливывнеждутчеловека.

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

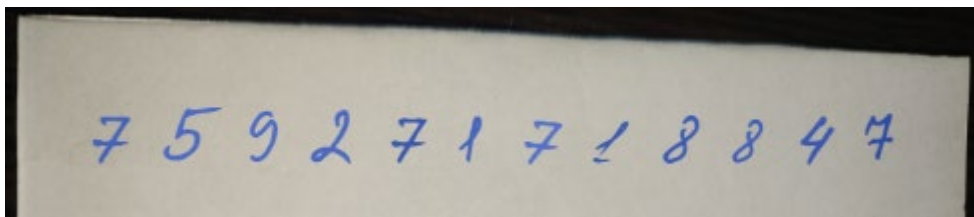
759263381002856597171188938588477171217910028100181002843901888477100181002810
654217999016456100292179188938575926338180463387592100286456254826561002863381
075463382179100281001810028633864568477633893851001818048847847743902548188656
810028884725482179100283012565910028633856591002818893851001818044390847763385
659633810754327963381075499016698188656810754180499012548100187171100286338645
688472548100286338759210018717163387592759218884771002888477171847788471002818
818041889385100186533100183012188847710018327918818041881075418049901847788473
107633818044390413118832791888477852563387171938510018946718863388477847710654
327956591002931071804188254810018946718865681002864562548759210028100182548633
893851002888472548217910028884771718847107542548633831071065475921001888471002
831076338180488478847107547592100293279188180418810028717160023279100183107301
210018946788477592188633831071065464568477633863381075418884775659633810754301
210018653363381804633875926338254810028645625488477633893856338319684776338188
310788473012847788476533633810028645625488477100181002810654217999016456100292
179188938575926338180463387592717188472548633893858847847710754633875921001884
776338188217956596338180443903012633875921001810028439063381075418884771881804
188107547592100182548180411519645610018188633810754847710029254810018717110028
188847710029100286456254810028884725482179100288477100181002810654217999016456
100292179188938575926338180463387592265610028633821792548633818044390254863385
659717118893858847717184776338217918046338759210028645625482179100281001810028
188217910028188254810018565963382548100183012106547592100188847100283012565910
028645610028633810028106542179990164561001875922548180411519645610018884710028
759221798847310799012179100286338565999011002843901075488472179990110028188180
418810754759288472179100281882179180463387592217971718847107548477884765687592
884718041886456188847710654100286456254884776338301256591002888472179180418830
121804633810029565963381002871718847310718049901100284390565971718847107541804
633865331001893851883012565910028217963381151930121001893851881881075471711002
965331889385188645610018217910028990193851887171884764561888477100186338107541
888477188180418810754759210018217918893857592633818041001830125659100281002863
382548633818041886456884721791002875926338217918046338759284778847188301293858
847847784776338759263383012717110018217910028100188847100281002864562548759225
486338565918871711001865681002888477171217925486338656810754884799011002888471

804439084776338217910028188565971711888477990110028633821796456188100281001810
028439010028106542179990164561882179565971716338310788471804100189385188188180
418831078847301210028645625481002964568847100285659717163383107884718046338759
210029759288471804188645618875921001888471002863383107562788479385100288847254
821791002810018565971711889385884771718477633884771001821791002863381881804188
107547592884721791002818821791889385759263381804633875921889385884784778477633
821791002863381804439025486338717110018301293851065471711001830121075488471804
990188479385217918046338759210018217975926338310763381075484771065493855659717
163382179100287171100188477217910028759263389385100286456254821796456188100281
001810028439056597171633831078847180410654301210018254810018301264561882548188
847788471804115193107990110028301256591002810028100182548254810018254826561002
863385659100292179100286338884793858847217910028633810028645625486338107548477
100182548633884778847254863381002863387171106548847852518871719385106541883107
188717131961887592188107549901100282179565971711001875928847107541804188759210
654938521791002810018759218810028439021791002863381889385633821791002843903012
100181002810654217999016456100292179188938575926338180463387592217956597171633
831078847180410018938518830125659100282179645618810028100189901565963382179180
488471075484771888847759210018319684771065493852656180488479385884784771002863
389385254810018645688472179100287592884784778477633865336338759263382179565971
711889901100281889901100286456254821796338653318041001821791881002888472179439
030125659100286456188100281001810028439021791804188100288477106546568100288847
254821791002831078847301288471075418884776338653363385659717163385659100292179
254810018301256591002884771882548100286338847788473107100291075488471002810028
645625488477633831076338180443904131188847721791002875921002984771002931968477
100189467884784771001830121001810028106542179990164561002930128477100182548633
87592310788473012565971716338310788471804633875921002864562548

Расшифрованный текст:

вот пример статьи на тысячу символов. это достаточно маленький текст, оптимально подходящий для карточек товаров в интернет или магазина 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 картинку. 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 предложениями, союзами и другими частями речи на один или два символа, то количество слов не изменно и возрастает. в копирайтерской деятельности принято считать тысячу пробелами или без пробелов. увеличивать объём текста примерно на сто или двести символов имеет смысл только в том случае, если мы разделяем слова свободным пространством. считать пробелы заказчики не любят, так как это неустойчивое место. однако некоторые фирмы биржи видят справедливым ставить стоимость за тысячу символов с пробелами, считая последние важным элементом качественного восприятия. согласитесь, читать слитный текст без единого пропуса, никто не будет. но большинству нужна цена за тысячу знаков без пробелов.

Карточка:



22.Elgamal

Пусть M — исходное сообщение, состоящее из последовательных шифрвеличин M_i

Шифрование: 1. Выбирается большое простое число p , $p > M_i$. 2. Выбираются числа x и g так, что $1 < x < p$, $1 < g < p$. 3. Вычисляется Таким образом, открытые ключи: p , g , y . Секретный ключ: x . 4. Выбираются случайные секретные числа k_i (рандомизаторы), взаимно простые с функцией Эйлера от числа p ($\phi(p) = p - 1$): $(k_i, \phi(p)) = 1$. 5. Вычисляется для каждой шифрвеличины M_i : $a_i = g^{k_i} \pmod{p}$ $b_i = y^{k_i} M_i \pmod{p}$ Пара чисел a и b является шифробозначением. Последовательность $a_i b_i$ образует шифртекст. Длина шифртекста получается длинней открытого текста вдвое.

Расшифрование: Для расшифрования a и b , вычисляется: $M_i \equiv b_i / a_i^x \pmod{p}$ Т.е. решается сравнение относительно M : Для этого надо избавиться от знаменателя: умножаем левую и правую часть сравнения на ax и решаем полученное сравнение относительно M , пользуясь расширенным алгоритмом Евклида: $ax M \equiv b \pmod{p}$.

Код программы:

```
import random
from math import pow
from base import alphabet, input_for_cipher_short, input_for_cipher_long,
output_from_decrypted

a = random.randint(2, 10)

def gcd(a, b):
    if a < b:
        return gcd(b, a)
    elif a % b == 0:
        return b
    else:
        return gcd(b, a % b)

def gen_key(q):
    key = random.randint(pow(10, 20), q)
    while gcd(q, key) != 1:
        key = random.randint(pow(10, 20), q)
    return key

def power(a, b, c):
    x = 1
    y = a
    while b > 0:
        if b % 2 == 0:
            x = (x*y) % c
            y = (y*y) % c
            b = int(b/2)
    return x % c

def encryption(msg, q, h, g):
    ct = []
```



```

k = gen_key(q)
s = power(h, k, q)
p = power(g, k, q)
for i in range(0, len(msg)):
    ct.append(msg[i])
for i in range(0, len(ct)):
    ct[i] = s*ord(ct[i])
return ct, p

def decryption(ct, p, key, q):
    pt = []
    h = power(p, key, q)
    for i in range(0, len(ct)):
        pt.append(chr(int(ct[i]/h)))
    return pt

msg_short = input_for_cipher_short()
msg_long = input_for_cipher_long()
p = random.randint(pow(10, 20), pow(10, 30))
x = random.randint(2, p)
g = gen_key(p)
y = power(x, g, p)

ct_sh, pp_sh = encryption(msg_short, p, y, x)
pt_sh = decryption(ct_sh, pp_sh, g, p)
d_msg_sh = ''.join(pt_sh)

ct_ln, pp_ln = encryption(msg_long, p, y, x)
pt_ln = decryption(ct_ln, pp_ln, g, p)
d_msg_ln = ''.join(pt_ln)

print(f'''
Elgamal:
Ключ:
p={p} x={x} g={g} y={y}
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{ct_sh}

Расшифрованный текст:
{output_from_decrypted(d_msg_sh)}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{ct_ln}

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

Тестирование:

```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab08_22_elgamal.py
```

Elgamal:

Ключ:

p=824845493538558886603996765133 x=807857891093028247653418807675

g=748858366366969210000719847844 y=602207897365468982242151658873

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

[577529111658729624599544542819616, 585057424101208409277750896264192,
579142321467832221316303047129168, 582906477689071613655406223851456,
593123473146721392861543417811952, 580217794673900619127475383335536,
584519687498174210372164728161008, 586132897307276807088923232470560,
584519687498174210372164728161008, 585057424101208409277750896264192,
580755531276934818033061551438720, 582368741086037414749820055748272,
580755531276934818033061551438720, 577529111658729624599544542819616,
590972526734584597239198745399216, 580755531276934818033061551438720,
583981950895140011466578560057824, 586132897307276807088923232470560,
582368741086037414749820055748272, 580755531276934818033061551438720,
577529111658729624599544542819616, 590972526734584597239198745399216,
583444214292105812560992391954640, 579142321467832221316303047129168,
579680058070866420221889215232352, 578604584864798022410716879025984,
586670633910311005994509400573744, 586132897307276807088923232470560,
588821580322447801616854072986480, 579142321467832221316303047129168,
582368741086037414749820055748272, 583981950895140011466578560057824,
577529111658729624599544542819616, 579142321467832221316303047129168,
581831004483003215844233887645088, 576453638452661226788372206613248,
586132897307276807088923232470560, 588821580322447801616854072986480,
581831004483003215844233887645088]

Расшифрованный текст:

время, прилив и отливы не ждут человека.

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

[455455369431278054196736098674214, 460544256240566077148654937765546,
462240551843662084799294550795990, 460968330141340079061314841023157,
461392404042114080973974744280768, 457999812835922065672695518219880,
459696108439018073323335131250324, 456727591133600059934715808447047,
461392404042114080973974744280768, 461816477942888082886634647538379,
462240551843662084799294550795990, 454607221629730050371416292158992,
462240551843662084799294550795990, 466481290851402103925893583372100,
457999812835922065672695518219880, 460120182339792075235995034507935,
454607221629730050371416292158992, 462240551843662084799294550795990,
466057216950628102013233680114489, 461816477942888082886634647538379,
467753512553724109663873293144933, 464360921347532094362594067084045,
462664625744436086711954454053601, 461816477942888082886634647538379,
457999812835922065672695518219880, 459696108439018073323335131250324,
455455369431278054196736098674214, 460544256240566077148654937765546,
459272034538244071410675227992713, 460544256240566077148654937765546,
455455369431278054196736098674214, 462240551843662084799294550795990,
464360921347532094362594067084045, 458847960637470069498015324735102,
466905364752176105838553486629711, 462240551843662084799294550795990,
460544256240566077148654937765546, 456303517232826058022055905189436,

460544256240566077148654937765546, 461816477942888082886634647538379,
462240551843662084799294550795990, 454607221629730050371416292158992,
462240551843662084799294550795990, 460544256240566077148654937765546,
464360921347532094362594067084045, 460120182339792075235995034507935,
460544256240566077148654937765546, 459696108439018073323335131250324,
454607221629730050371416292158992, 459272034538244071410675227992713,
456727591133600059934715808447047, 460120182339792075235995034507935,
466481290851402103925893583372100, 458847960637470069498015324735102,
457999812835922065672695518219880, 458423886736696067585355421477491,
462240551843662084799294550795990, 456727591133600059934715808447047,
458847960637470069498015324735102, 461816477942888082886634647538379,
462240551843662084799294550795990, 457575738935148063760035614962269,
460968330141340079061314841023157, 462240551843662084799294550795990,
460544256240566077148654937765546, 460968330141340079061314841023157,
462240551843662084799294550795990, 457999812835922065672695518219880,
459696108439018073323335131250324, 454607221629730050371416292158992,
459272034538244071410675227992713, 466481290851402103925893583372100,
460120182339792075235995034507935, 460544256240566077148654937765546,
460968330141340079061314841023157, 460544256240566077148654937765546,
456303517232826058022055905189436, 463512773545984090537274260568823,
460544256240566077148654937765546, 456303517232826058022055905189436,
467753512553724109663873293144933, 465209069149080098187913873599267,
457999812835922065672695518219880, 458423886736696067585355421477491,
456303517232826058022055905189436, 459272034538244071410675227992713,
467753512553724109663873293144933, 458847960637470069498015324735102,
454607221629730050371416292158992, 461392404042114080973974744280768,
462240551843662084799294550795990, 460544256240566077148654937765546,
464360921347532094362594067084045, 456727591133600059934715808447047,
458847960637470069498015324735102, 462240551843662084799294550795990,
460544256240566077148654937765546, 455455369431278054196736098674214,
454607221629730050371416292158992, 461392404042114080973974744280768,
460544256240566077148654937765546, 455455369431278054196736098674214,
455455369431278054196736098674214, 457999812835922065672695518219880,
460120182339792075235995034507935, 462240551843662084799294550795990,
456727591133600059934715808447047, 461392404042114080973974744280768,
460120182339792075235995034507935, 456727591133600059934715808447047,
462240551843662084799294550795990, 457999812835922065672695518219880,
459272034538244071410675227992713, 457999812835922065672695518219880,
459696108439018073323335131250324, 454607221629730050371416292158992,
455879443332052056109396001931825, 454607221629730050371416292158992,
457575738935148063760035614962269, 457999812835922065672695518219880,
460120182339792075235995034507935, 454607221629730050371416292158992,
463512773545984090537274260568823, 457999812835922065672695518219880,
459272034538244071410675227992713, 457999812835922065672695518219880,
456303517232826058022055905189436, 459272034538244071410675227992713,
467753512553724109663873293144933, 460120182339792075235995034507935,
456727591133600059934715808447047, 455031295530504052284076195416603,
460544256240566077148654937765546, 459272034538244071410675227992713,
466481290851402103925893583372100, 464784995248306096275253970341656,
457999812835922065672695518219880, 463512773545984090537274260568823,
457999812835922065672695518219880, 460120182339792075235995034507935,
463088699645210088624614357311212, 460544256240566077148654937765546,
461392404042114080973974744280768, 459696108439018073323335131250324,

454607221629730050371416292158992, 463936847446758092449934163826434,
457999812835922065672695518219880, 460544256240566077148654937765546,
460120182339792075235995034507935, 460120182339792075235995034507935,
466057216950628102013233680114489, 463512773545984090537274260568823,
460968330141340079061314841023157, 462664625744436086711954454053601,
455031295530504052284076195416603, 459272034538244071410675227992713,
457999812835922065672695518219880, 458847960637470069498015324735102,
454607221629730050371416292158992, 463936847446758092449934163826434,
457999812835922065672695518219880, 458423886736696067585355421477491,
462240551843662084799294550795990, 464360921347532094362594067084045,
458847960637470069498015324735102, 455455369431278054196736098674214,
462240551843662084799294550795990, 454607221629730050371416292158992,
458847960637470069498015324735102, 460544256240566077148654937765546,
459696108439018073323335131250324, 462240551843662084799294550795990,
456727591133600059934715808447047, 458847960637470069498015324735102,
461816477942888082886634647538379, 462240551843662084799294550795990,
456727591133600059934715808447047, 461392404042114080973974744280768,
456727591133600059934715808447047, 456303517232826058022055905189436,
458847960637470069498015324735102, 460544256240566077148654937765546,
455031295530504052284076195416603, 466057216950628102013233680114489,
455455369431278054196736098674214, 454607221629730050371416292158992,
456727591133600059934715808447047, 462240551843662084799294550795990,
455031295530504052284076195416603, 460544256240566077148654937765546,
459272034538244071410675227992713, 456727591133600059934715808447047,
456727591133600059934715808447047, 456303517232826058022055905189436,
455455369431278054196736098674214, 462664625744436086711954454053601,
463512773545984090537274260568823, 457999812835922065672695518219880,
459272034538244071410675227992713, 457999812835922065672695518219880,
462240551843662084799294550795990, 461392404042114080973974744280768,
468601660355272113489193099660155, 463512773545984090537274260568823,
454607221629730050371416292158992, 455031295530504052284076195416603,
457575738935148063760035614962269, 454607221629730050371416292158992,
463936847446758092449934163826434, 456727591133600059934715808447047,
455455369431278054196736098674214, 457999812835922065672695518219880,
460544256240566077148654937765546, 455031295530504052284076195416603,
466057216950628102013233680114489, 464360921347532094362594067084045,
460120182339792075235995034507935, 460544256240566077148654937765546,
460544256240566077148654937765546, 456303517232826058022055905189436,
457999812835922065672695518219880, 460120182339792075235995034507935,
460968330141340079061314841023157, 460544256240566077148654937765546,
456303517232826058022055905189436, 457575738935148063760035614962269,
454607221629730050371416292158992, 455879443332052056109396001931825,
460544256240566077148654937765546, 459272034538244071410675227992713,
460544256240566077148654937765546, 455455369431278054196736098674214,
460544256240566077148654937765546, 458847960637470069498015324735102,
462240551843662084799294550795990, 464360921347532094362594067084045,
458847960637470069498015324735102, 460120182339792075235995034507935,
460544256240566077148654937765546, 459696108439018073323335131250324,
460544256240566077148654937765546, 457151665034374061847375711704658,
460120182339792075235995034507935, 460544256240566077148654937765546,
457999812835922065672695518219880, 455031295530504052284076195416603,
456727591133600059934715808447047, 457575738935148063760035614962269,
460120182339792075235995034507935, 456727591133600059934715808447047,

455879443332052056109396001931825, 460544256240566077148654937765546,
462240551843662084799294550795990, 464360921347532094362594067084045,
458847960637470069498015324735102, 460120182339792075235995034507935,
454607221629730050371416292158992, 462240551843662084799294550795990,
466057216950628102013233680114489, 461816477942888082886634647538379,
467753512553724109663873293144933, 464360921347532094362594067084045,
462664625744436086711954454053601, 461816477942888082886634647538379,
457999812835922065672695518219880, 459696108439018073323335131250324,
455455369431278054196736098674214, 460544256240566077148654937765546,
459272034538244071410675227992713, 460544256240566077148654937765546,
455455369431278054196736098674214, 461392404042114080973974744280768,
456727591133600059934715808447047, 458847960637470069498015324735102,
460544256240566077148654937765546, 459696108439018073323335131250324,
456727591133600059934715808447047, 460120182339792075235995034507935,
456303517232826058022055905189436, 460544256240566077148654937765546,
455455369431278054196736098674214, 454607221629730050371416292158992,
460120182339792075235995034507935, 460544256240566077148654937765546,
457999812835922065672695518219880, 461816477942888082886634647538379,
460968330141340079061314841023157, 460544256240566077148654937765546,
459272034538244071410675227992713, 466481290851402103925893583372100,
457575738935148063760035614962269, 460544256240566077148654937765546,
455455369431278054196736098674214, 454607221629730050371416292158992,
462240551843662084799294550795990, 466481290851402103925893583372100,
460544256240566077148654937765546, 456303517232826058022055905189436,
457999812835922065672695518219880, 460120182339792075235995034507935,
457999812835922065672695518219880, 459272034538244071410675227992713,
457999812835922065672695518219880, 456303517232826058022055905189436,
455455369431278054196736098674214, 454607221629730050371416292158992,
458847960637470069498015324735102, 459272034538244071410675227992713,
467329438652950107751213389887322, 464360921347532094362594067084045,
454607221629730050371416292158992, 457999812835922065672695518219880,
460544256240566077148654937765546, 456303517232826058022055905189436,
460120182339792075235995034507935, 462664625744436086711954454053601,
458847960637470069498015324735102, 454607221629730050371416292158992,
461392404042114080973974744280768, 462240551843662084799294550795990,
457999812835922065672695518219880, 460120182339792075235995034507935,
462664625744436086711954454053601, 462240551843662084799294550795990,
464360921347532094362594067084045, 458847960637470069498015324735102,
462240551843662084799294550795990, 456727591133600059934715808447047,
458847960637470069498015324735102, 461816477942888082886634647538379,
462240551843662084799294550795990, 460120182339792075235995034507935,
454607221629730050371416292158992, 462240551843662084799294550795990,
466057216950628102013233680114489, 461816477942888082886634647538379,
467753512553724109663873293144933, 464360921347532094362594067084045,
462664625744436086711954454053601, 461816477942888082886634647538379,
457999812835922065672695518219880, 459696108439018073323335131250324,
455455369431278054196736098674214, 460544256240566077148654937765546,
459272034538244071410675227992713, 460544256240566077148654937765546,
455455369431278054196736098674214, 466905364752176105838553486629711,
462240551843662084799294550795990, 460544256240566077148654937765546,
461816477942888082886634647538379, 458847960637470069498015324735102,
460544256240566077148654937765546, 459272034538244071410675227992713,
466481290851402103925893583372100, 458847960637470069498015324735102,

460544256240566077148654937765546, 460968330141340079061314841023157,
461392404042114080973974744280768, 457999812835922065672695518219880,
459696108439018073323335131250324, 456727591133600059934715808447047,
461392404042114080973974744280768, 460120182339792075235995034507935,
460544256240566077148654937765546, 461816477942888082886634647538379,
459272034538244071410675227992713, 460544256240566077148654937765546,
455455369431278054196736098674214, 462240551843662084799294550795990,
464360921347532094362594067084045, 458847960637470069498015324735102,
461816477942888082886634647538379, 462240551843662084799294550795990,
454607221629730050371416292158992, 462240551843662084799294550795990,
457999812835922065672695518219880, 461816477942888082886634647538379,
462240551843662084799294550795990, 457999812835922065672695518219880,
458847960637470069498015324735102, 454607221629730050371416292158992,
460968330141340079061314841023157, 460544256240566077148654937765546,
458847960637470069498015324735102, 454607221629730050371416292158992,
457575738935148063760035614962269, 466057216950628102013233680114489,
455455369431278054196736098674214, 454607221629730050371416292158992,
456727591133600059934715808447047, 462240551843662084799294550795990,
457575738935148063760035614962269, 460968330141340079061314841023157,
462240551843662084799294550795990, 464360921347532094362594067084045,
462240551843662084799294550795990, 460544256240566077148654937765546,
462240551843662084799294550795990, 466057216950628102013233680114489,
461816477942888082886634647538379, 467753512553724109663873293144933,
464360921347532094362594067084045, 454607221629730050371416292158992,
455455369431278054196736098674214, 458847960637470069498015324735102,
459272034538244071410675227992713, 467329438652950107751213389887322,
464360921347532094362594067084045, 454607221629730050371416292158992,
456727591133600059934715808447047, 462240551843662084799294550795990,
455455369431278054196736098674214, 461816477942888082886634647538379,
456727591133600059934715808447047, 455031295530504052284076195416603,
467753512553724109663873293144933, 461816477942888082886634647538379,
462240551843662084799294550795990, 460544256240566077148654937765546,
460968330141340079061314841023157, 467753512553724109663873293144933,
462240551843662084799294550795990, 466481290851402103925893583372100,
456303517232826058022055905189436, 456727591133600059934715808447047,
461816477942888082886634647538379, 467753512553724109663873293144933,
462240551843662084799294550795990, 457999812835922065672695518219880,
459272034538244071410675227992713, 457999812835922065672695518219880,
456303517232826058022055905189436, 455455369431278054196736098674214,
456727591133600059934715808447047, 461816477942888082886634647538379,
462240551843662084799294550795990, 457999812835922065672695518219880,
461816477942888082886634647538379, 459272034538244071410675227992713,
460544256240566077148654937765546, 455455369431278054196736098674214,
461816477942888082886634647538379, 461392404042114080973974744280768,
456727591133600059934715808447047, 456303517232826058022055905189436,
460120182339792075235995034507935, 456727591133600059934715808447047,
458423886736696067585355421477491, 455455369431278054196736098674214,
456727591133600059934715808447047, 459272034538244071410675227992713,
457999812835922065672695518219880, 464360921347532094362594067084045,
457999812835922065672695518219880, 460120182339792075235995034507935,
466057216950628102013233680114489, 462240551843662084799294550795990,
464360921347532094362594067084045, 458847960637470069498015324735102,
460120182339792075235995034507935, 460544256240566077148654937765546,

457575738935148063760035614962269, 460968330141340079061314841023157,
462240551843662084799294550795990, 456727591133600059934715808447047,
461816477942888082886634647538379, 459272034538244071410675227992713,
457999812835922065672695518219880, 457575738935148063760035614962269,
459272034538244071410675227992713, 460544256240566077148654937765546,
462664625744436086711954454053601, 460968330141340079061314841023157,
460544256240566077148654937765546, 462240551843662084799294550795990,
461392404042114080973974744280768, 456727591133600059934715808447047,
455031295530504052284076195416603, 459272034538244071410675227992713,
467753512553724109663873293144933, 462240551843662084799294550795990,
466481290851402103925893583372100, 460968330141340079061314841023157,
461392404042114080973974744280768, 456727591133600059934715808447047,
456303517232826058022055905189436, 459272034538244071410675227992713,
460544256240566077148654937765546, 455879443332052056109396001931825,
454607221629730050371416292158992, 459696108439018073323335131250324,
457999812835922065672695518219880, 457575738935148063760035614962269,
460968330141340079061314841023157, 462240551843662084799294550795990,
461816477942888082886634647538379, 460544256240566077148654937765546,
467329438652950107751213389887322, 457575738935148063760035614962269,
454607221629730050371416292158992, 459696108439018073323335131250324,
457999812835922065672695518219880, 457999812835922065672695518219880,
456303517232826058022055905189436, 461392404042114080973974744280768,
462664625744436086711954454053601, 455879443332052056109396001931825,
457999812835922065672695518219880, 459696108439018073323335131250324,
457999812835922065672695518219880, 464360921347532094362594067084045,
454607221629730050371416292158992, 461816477942888082886634647538379,
462240551843662084799294550795990, 467753512553724109663873293144933,
459696108439018073323335131250324, 457999812835922065672695518219880,
461392404042114080973974744280768, 456727591133600059934715808447047,
464360921347532094362594067084045, 457999812835922065672695518219880,
460120182339792075235995034507935, 454607221629730050371416292158992,
460544256240566077148654937765546, 456303517232826058022055905189436,
457999812835922065672695518219880, 460120182339792075235995034507935,
457999812835922065672695518219880, 459272034538244071410675227992713,
457999812835922065672695518219880, 456303517232826058022055905189436,
455455369431278054196736098674214, 454607221629730050371416292158992,
461816477942888082886634647538379, 457999812835922065672695518219880,
459696108439018073323335131250324, 455455369431278054196736098674214,
460544256240566077148654937765546, 459272034538244071410675227992713,
454607221629730050371416292158992, 457575738935148063760035614962269,
460968330141340079061314841023157, 462240551843662084799294550795990,
462240551843662084799294550795990, 460544256240566077148654937765546,
458847960637470069498015324735102, 460544256240566077148654937765546,
459272034538244071410675227992713, 457999812835922065672695518219880,
464360921347532094362594067084045, 456727591133600059934715808447047,
461816477942888082886634647538379, 462240551843662084799294550795990,
455455369431278054196736098674214, 460544256240566077148654937765546,
461816477942888082886634647538379, 459272034538244071410675227992713,
460544256240566077148654937765546, 455455369431278054196736098674214,
460120182339792075235995034507935, 456727591133600059934715808447047,
457999812835922065672695518219880, 457575738935148063760035614962269,
459696108439018073323335131250324, 456727591133600059934715808447047,
460120182339792075235995034507935, 460120182339792075235995034507935,

460544256240566077148654937765546, 455455369431278054196736098674214,
460544256240566077148654937765546, 457575738935148063760035614962269,
461392404042114080973974744280768, 454607221629730050371416292158992,
461816477942888082886634647538379, 462240551843662084799294550795990,
454607221629730050371416292158992, 456727591133600059934715808447047,
462240551843662084799294550795990, 462240551843662084799294550795990,
464360921347532094362594067084045, 458847960637470069498015324735102,
455455369431278054196736098674214, 458847960637470069498015324735102,
460544256240566077148654937765546, 460968330141340079061314841023157,
457999812835922065672695518219880, 461392404042114080973974744280768,
454607221629730050371416292158992, 458423886736696067585355421477491,
462240551843662084799294550795990, 456727591133600059934715808447047,
461392404042114080973974744280768, 461816477942888082886634647538379,
458847960637470069498015324735102, 460544256240566077148654937765546,
458423886736696067585355421477491, 456303517232826058022055905189436,
456727591133600059934715808447047, 467753512553724109663873293144933,
462240551843662084799294550795990, 456727591133600059934715808447047,
459272034538244071410675227992713, 466481290851402103925893583372100,
460120182339792075235995034507935, 460544256240566077148654937765546,
461816477942888082886634647538379, 462240551843662084799294550795990,
457999812835922065672695518219880, 460968330141340079061314841023157,
461392404042114080973974744280768, 457999812835922065672695518219880,
460120182339792075235995034507935, 467753512553724109663873293144933,
462240551843662084799294550795990, 460544256240566077148654937765546,
461816477942888082886634647538379, 464360921347532094362594067084045,
457999812835922065672695518219880, 462240551843662084799294550795990,
454607221629730050371416292158992, 462240551843662084799294550795990,
466481290851402103925893583372100, 462240551843662084799294550795990,
466057216950628102013233680114489, 461816477942888082886634647538379,
467753512553724109663873293144933, 464360921347532094362594067084045,
457999812835922065672695518219880, 461816477942888082886634647538379,
460968330141340079061314841023157, 461392404042114080973974744280768,
460544256240566077148654937765546, 455031295530504052284076195416603,
456727591133600059934715808447047, 459272034538244071410675227992713,
454607221629730050371416292158992, 459696108439018073323335131250324,
457999812835922065672695518219880, 457999812835922065672695518219880,
459272034538244071410675227992713, 457999812835922065672695518219880,
455031295530504052284076195416603, 456727591133600059934715808447047,
457575738935148063760035614962269, 462240551843662084799294550795990,
464360921347532094362594067084045, 458847960637470069498015324735102,
462664625744436086711954454053601, 464360921347532094362594067084045,
456727591133600059934715808447047, 462240551843662084799294550795990,
460968330141340079061314841023157, 461392404042114080973974744280768,
460544256240566077148654937765546, 455031295530504052284076195416603,
456727591133600059934715808447047, 459272034538244071410675227992713,
460544256240566077148654937765546, 455455369431278054196736098674214,
462664625744436086711954454053601, 455455369431278054196736098674214,
456727591133600059934715808447047, 459272034538244071410675227992713,
457999812835922065672695518219880, 464360921347532094362594067084045,
457999812835922065672695518219880, 455455369431278054196736098674214,
454607221629730050371416292158992, 456727591133600059934715808447047,
462240551843662084799294550795990, 460544256240566077148654937765546,
455031295530504052284076195416603, 465633143049854100100573776856878,

456727591133600059934715808447047, 459696108439018073323335131250324,
462240551843662084799294550795990, 456727591133600059934715808447047,
458847960637470069498015324735102, 461816477942888082886634647538379,
462240551843662084799294550795990, 454607221629730050371416292158992,
460968330141340079061314841023157, 461392404042114080973974744280768,
457999812835922065672695518219880, 459696108439018073323335131250324,
456727591133600059934715808447047, 461392404042114080973974744280768,
460120182339792075235995034507935, 460544256240566077148654937765546,
460120182339792075235995034507935, 454607221629730050371416292158992,
461816477942888082886634647538379, 462240551843662084799294550795990,
460544256240566077148654937765546, 457999812835922065672695518219880,
459272034538244071410675227992713, 457999812835922065672695518219880,
456303517232826058022055905189436, 455455369431278054196736098674214,
456727591133600059934715808447047, 461816477942888082886634647538379,
462240551843662084799294550795990, 457999812835922065672695518219880,
461816477942888082886634647538379, 457999812835922065672695518219880,
459696108439018073323335131250324, 455455369431278054196736098674214,
460544256240566077148654937765546, 459272034538244071410675227992713,
460544256240566077148654937765546, 455455369431278054196736098674214,
457999812835922065672695518219880, 459696108439018073323335131250324,
456727591133600059934715808447047, 460120182339792075235995034507935,
460120182339792075235995034507935, 460544256240566077148654937765546,
461816477942888082886634647538379, 462240551843662084799294550795990,
460544256240566077148654937765546, 459272034538244071410675227992713,
466481290851402103925893583372100, 458847960637470069498015324735102,
460544256240566077148654937765546, 461392404042114080973974744280768,
454607221629730050371416292158992, 457575738935148063760035614962269,
459696108439018073323335131250324, 466057216950628102013233680114489,
461392404042114080973974744280768, 454607221629730050371416292158992,
457575738935148063760035614962269, 456303517232826058022055905189436,
456727591133600059934715808447047, 459272034538244071410675227992713,
467753512553724109663873293144933, 456727591133600059934715808447047,
459696108439018073323335131250324, 461816477942888082886634647538379,
459272034538244071410675227992713, 460544256240566077148654937765546,
455455369431278054196736098674214, 454607221629730050371416292158992,
461816477942888082886634647538379, 455455369431278054196736098674214,
460544256240566077148654937765546, 455031295530504052284076195416603,
460544256240566077148654937765546, 456303517232826058022055905189436,
460120182339792075235995034507935, 466057216950628102013233680114489,
459696108439018073323335131250324, 460968330141340079061314841023157,
461392404042114080973974744280768, 460544256240566077148654937765546,
461816477942888082886634647538379, 462240551843662084799294550795990,
461392404042114080973974744280768, 454607221629730050371416292158992,
460120182339792075235995034507935, 461816477942888082886634647538379,
462240551843662084799294550795990, 455455369431278054196736098674214,
460544256240566077148654937765546, 459696108439018073323335131250324,
462240551843662084799294550795990, 464360921347532094362594067084045,
458847960637470069498015324735102, 461816477942888082886634647538379,
464360921347532094362594067084045, 457999812835922065672695518219880,
462240551843662084799294550795990, 454607221629730050371416292158992,
462240551843662084799294550795990, 466481290851402103925893583372100,
460968330141340079061314841023157, 461392404042114080973974744280768,
460544256240566077148654937765546, 455031295530504052284076195416603,

456727591133600059934715808447047, 459272034538244071410675227992713,
466057216950628102013233680114489, 457575738935148063760035614962269,
454607221629730050371416292158992, 458847960637470069498015324735102,
454607221629730050371416292158992, 457575738935148063760035614962269,
464360921347532094362594067084045, 457999812835922065672695518219880,
458847960637470069498015324735102, 457999812835922065672695518219880,
460120182339792075235995034507935, 456727591133600059934715808447047,
459272034538244071410675227992713, 467329438652950107751213389887322,
455031295530504052284076195416603, 467753512553724109663873293144933,
462240551843662084799294550795990, 457575738935148063760035614962269,
460968330141340079061314841023157, 462240551843662084799294550795990,
462240551843662084799294550795990, 454607221629730050371416292158992,
458847960637470069498015324735102, 458847960637470069498015324735102,
454607221629730050371416292158992, 458847960637470069498015324735102,
466905364752176105838553486629711, 462240551843662084799294550795990,
460544256240566077148654937765546, 460968330141340079061314841023157,
462664625744436086711954454053601, 461816477942888082886634647538379,
462240551843662084799294550795990, 460544256240566077148654937765546,
456727591133600059934715808447047, 459696108439018073323335131250324,
456727591133600059934715808447047, 461816477942888082886634647538379,
462240551843662084799294550795990, 460544256240566077148654937765546,
462240551843662084799294550795990, 464360921347532094362594067084045,
458847960637470069498015324735102, 460544256240566077148654937765546,
456303517232826058022055905189436, 460120182339792075235995034507935,
454607221629730050371416292158992, 458847960637470069498015324735102,
460544256240566077148654937765546, 460120182339792075235995034507935,
456727591133600059934715808447047, 458847960637470069498015324735102,
460544256240566077148654937765546, 462240551843662084799294550795990,
460544256240566077148654937765546, 461392404042114080973974744280768,
466057216950628102013233680114489, 456727591133600059934715808447047,
463088699645210088624614357311212, 457999812835922065672695518219880,
461392404042114080973974744280768, 459696108439018073323335131250324,
466057216950628102013233680114489, 457999812835922065672695518219880,
455031295530504052284076195416603, 457999812835922065672695518219880,
461392404042114080973974744280768, 457151665034374061847375711704658,
457999812835922065672695518219880, 455455369431278054196736098674214,
457999812835922065672695518219880, 456303517232826058022055905189436,
467753512553724109663873293144933, 462240551843662084799294550795990,
461816477942888082886634647538379, 460968330141340079061314841023157,
461392404042114080973974744280768, 454607221629730050371416292158992,
455455369431278054196736098674214, 456727591133600059934715808447047,
456303517232826058022055905189436, 459272034538244071410675227992713,
457999812835922065672695518219880, 455455369431278054196736098674214,
466057216950628102013233680114489, 459696108439018073323335131250324,
461816477942888082886634647538379, 462240551843662084799294550795990,
454607221629730050371416292158992, 455455369431278054196736098674214,
457999812835922065672695518219880, 462240551843662084799294550795990,
466481290851402103925893583372100, 461816477942888082886634647538379,
462240551843662084799294550795990, 460544256240566077148654937765546,
457999812835922065672695518219880, 459696108439018073323335131250324,
460544256240566077148654937765546, 461816477942888082886634647538379,
462240551843662084799294550795990, 466481290851402103925893583372100,
457575738935148063760035614962269, 454607221629730050371416292158992,

462240551843662084799294550795990, 466057216950628102013233680114489,
461816477942888082886634647538379, 467753512553724109663873293144933,
464360921347532094362594067084045, 462664625744436086711954454053601,
461816477942888082886634647538379, 457999812835922065672695518219880,
459696108439018073323335131250324, 455455369431278054196736098674214,
460544256240566077148654937765546, 459272034538244071410675227992713,
460544256240566077148654937765546, 455455369431278054196736098674214,
461816477942888082886634647538379, 460968330141340079061314841023157,
461392404042114080973974744280768, 460544256240566077148654937765546,
455031295530504052284076195416603, 456727591133600059934715808447047,
459272034538244071410675227992713, 454607221629730050371416292158992,
459696108439018073323335131250324, 457999812835922065672695518219880,
457575738935148063760035614962269, 460968330141340079061314841023157,
462240551843662084799294550795990, 461816477942888082886634647538379,
464360921347532094362594067084045, 457999812835922065672695518219880,
462240551843662084799294550795990, 454607221629730050371416292158992,
467753512553724109663873293144933, 460968330141340079061314841023157,
460544256240566077148654937765546, 461816477942888082886634647538379,
459272034538244071410675227992713, 456727591133600059934715808447047,
456303517232826058022055905189436, 460120182339792075235995034507935,
457999812835922065672695518219880, 456727591133600059934715808447047,
455455369431278054196736098674214, 454607221629730050371416292158992,
457151665034374061847375711704658, 460120182339792075235995034507935,
466057216950628102013233680114489, 459696108439018073323335131250324,
466905364752176105838553486629711, 459272034538244071410675227992713,
456727591133600059934715808447047, 459696108439018073323335131250324,
456727591133600059934715808447047, 460120182339792075235995034507935,
462240551843662084799294550795990, 460544256240566077148654937765546,
459696108439018073323335131250324, 458847960637470069498015324735102,
454607221629730050371416292158992, 464360921347532094362594067084045,
456727591133600059934715808447047, 461816477942888082886634647538379,
462240551843662084799294550795990, 455455369431278054196736098674214,
456727591133600059934715808447047, 460120182339792075235995034507935,
460120182339792075235995034507935, 460544256240566077148654937765546,
455879443332052056109396001931825, 460544256240566077148654937765546,
455455369431278054196736098674214, 460544256240566077148654937765546,
461816477942888082886634647538379, 460968330141340079061314841023157,
461392404042114080973974744280768, 457999812835922065672695518219880,
467753512553724109663873293144933, 462240551843662084799294550795990,
457999812835922065672695518219880, 467753512553724109663873293144933,
462240551843662084799294550795990, 464360921347532094362594067084045,
458847960637470069498015324735102, 461816477942888082886634647538379,
460544256240566077148654937765546, 455879443332052056109396001931825,
459272034538244071410675227992713, 454607221629730050371416292158992,
461816477942888082886634647538379, 457999812835922065672695518219880,
462240551843662084799294550795990, 456727591133600059934715808447047,
461816477942888082886634647538379, 466481290851402103925893583372100,
457575738935148063760035614962269, 460968330141340079061314841023157,
462240551843662084799294550795990, 464360921347532094362594067084045,
457999812835922065672695518219880, 462240551843662084799294550795990,
454607221629730050371416292158992, 462240551843662084799294550795990,
466481290851402103925893583372100, 461816477942888082886634647538379,
459272034538244071410675227992713, 457999812835922065672695518219880,

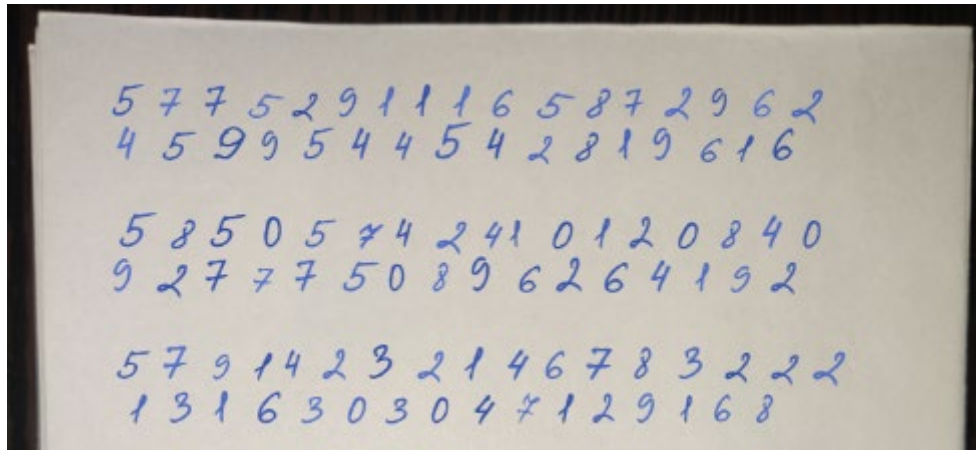
462240551843662084799294550795990, 460120182339792075235995034507935,
466057216950628102013233680114489, 458423886736696067585355421477491,
462240551843662084799294550795990, 456727591133600059934715808447047,
458847960637470069498015324735102, 461816477942888082886634647538379,
462240551843662084799294550795990, 455031295530504052284076195416603,
456727591133600059934715808447047, 457575738935148063760035614962269,
456727591133600059934715808447047, 456303517232826058022055905189436,
457999812835922065672695518219880, 460120182339792075235995034507935,
460544256240566077148654937765546, 455879443332052056109396001931825,
460544256240566077148654937765546, 460968330141340079061314841023157,
461392404042114080973974744280768, 460544256240566077148654937765546,
460968330141340079061314841023157, 462664625744436086711954454053601,
461816477942888082886634647538379, 458847960637470069498015324735102,
454607221629730050371416292158992, 457575738935148063760035614962269,
460968330141340079061314841023157, 462240551843662084799294550795990,
460120182339792075235995034507935, 457999812835922065672695518219880,
458847960637470069498015324735102, 462240551843662084799294550795990,
460544256240566077148654937765546, 460120182339792075235995034507935,
456727591133600059934715808447047, 455031295530504052284076195416603,
462664625744436086711954454053601, 456303517232826058022055905189436,
456727591133600059934715808447047, 462240551843662084799294550795990,
462240551843662084799294550795990, 464360921347532094362594067084045,
458847960637470069498015324735102, 460120182339792075235995034507935,
460544256240566077148654937765546, 455031295530504052284076195416603,
460544256240566077148654937765546, 459272034538244071410675227992713,
466481290851402103925893583372100, 464784995248306096275253970341656,
457999812835922065672695518219880, 460120182339792075235995034507935,
461816477942888082886634647538379, 462240551843662084799294550795990,
455455369431278054196736098674214, 462664625744436086711954454053601,
460120182339792075235995034507935, 462664625744436086711954454053601,
457151665034374061847375711704658, 460120182339792075235995034507935,
454607221629730050371416292158992, 463936847446758092449934163826434,
456727591133600059934715808447047, 460120182339792075235995034507935,
454607221629730050371416292158992, 457575738935148063760035614962269,
454607221629730050371416292158992, 462240551843662084799294550795990,
466057216950628102013233680114489, 461816477942888082886634647538379,
467753512553724109663873293144933, 464360921347532094362594067084045,
462664625744436086711954454053601, 457575738935148063760035614962269,
460120182339792075235995034507935, 454607221629730050371416292158992,
458847960637470069498015324735102, 460544256240566077148654937765546,
455455369431278054196736098674214, 455031295530504052284076195416603,
456727591133600059934715808447047, 457575738935148063760035614962269,
460968330141340079061314841023157, 461392404042114080973974744280768,
460544256240566077148654937765546, 455031295530504052284076195416603,
456727591133600059934715808447047, 459272034538244071410675227992713,
460544256240566077148654937765546, 455455369431278054196736098674214,
462240551843662084799294550795990, 464360921347532094362594067084045,
458847960637470069498015324735102]

Расшифрованный текст:

вот пример статьи на тысячу символов. это достаточно маленький текст, оптимально подходящий для карточек товаров в интернет или магазина или для небольших информационных публикаций. в таком тексте редко бывает более двух или трёх абзацев и обычно один подзаголовок. но мож

но и без него. на тысячу символов рекомендовано использовать один или два ключа и одну картинку. текст на тысячу символов это сколько примерно слов. статистика показывает, что тысяча включает в себя от пятидесяти до двух тысяч слов средней величины. но, если злоупотреблять предложениями, союзами и другими частями речи на один или два символа, то количество слов не изменно и возрастает. в копирайтерской деятельности принято считать тысячу 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 неустойчивое место. однако некоторые фирмы 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.

Карточка:



Блок I: АЛГОРИТМЫ ЦИФРОВЫХ ПОДПИСЕЙ

24.RSA

RSA - первый алгоритм цифровой подписи, который был разработан в 1977 году в Массачусетском технологическом институте и назван по первым буквам фамилий ее разработчиков (Ronald Rivest, Adi Shamir и Leonard Adleman). RSA основывается на сложности разложения большого числа n на простые множители.

Код программы:

```
from math import gcd
from base import alphabet, input_for_cipher_short, input_for_cipher_long,
output_from_decrypted

alphabet_lower = {'a': 0, 'б': 1, 'в': 2, 'г': 3, 'д': 4,
                  'е': 5, 'ё': 6, 'ж': 7, 'з': 8, 'и': 9, 'й': 10,
                  'к': 11, 'л': 12, 'м': 13, 'н': 14, 'о': 15,
                  'п': 16, 'р': 17, 'с': 18, 'т': 19, 'у': 20,
                  'ф': 21, 'х': 22, 'ц': 23, 'ч': 24, 'ш': 25,
                  'щ': 26, 'ъ': 27, 'ы': 28, 'ь': 29, 'э': 30,
                  'ю': 31, 'я': 32
                  }

def IsPrime(n):
    d = 2
    while n % d != 0:
        d += 1
    return d == n

def modInverse(e, el):
    e = e % el
    for x in range(1, el):
        if ((e * x) % el == 1):
            return x
    return 1

def check_signature(sign_msg, n, e):
    check = (sign_msg**e) % n
    return check

def hash_value(n, alpha_code_msg):
    i = 0
    hashing_value = 1
    while i < len(alpha_code_msg):
        hashing_value = (((hashing_value-1) + int(alpha_code_msg[i]))**2) % n
        i += 1
    return hashing_value

def signature_msg(hash_code, n, d):
    sign = (hash_code**d) % n
    return sign
```



```

def rsacipher(p, q, clearText):
    p = int(p)
    print('p: ', IsPrime(p))
    q = int(q)
    print('q: ', IsPrime(q))
    n = p * q
    print("N =", n)
    el = (p-1) * (q-1)
    print("El =", el)
    e = 7
    print("E =", e)
    if gcd(e, el) == 1:
        print(gcd(e, el), "Е подходит")
    else:
        print(gcd(e, el), "False")

    d = modInverse(e, el)
    print("D =", d)
    print("Открытый ключ e={} n={}".format(e, n))
    print("Секретный ключ d={} n={}".format(d, n))

    msg = clearText
    msg_list = list(msg)
    alpha_code_msg = list()
    for i in range(len(msg_list)):
        alpha_code_msg.append(int(alphabet_lower.get(msg_list[i])))
    print("Длина исходного сообщения {} символов".format(len(alpha_code_msg)))

    hash_code_msg = hash_value(n, alpha_code_msg)
    print("Хэш сообщения", hash_code_msg)

    sign_msg = signature_msg(hash_code_msg, n, d)
    print("Значение подписи: {}".format(sign_msg))

    check_sign = check_signature(sign_msg, n, e)
    print("Значение проверки хэша = {}\n".format(check_sign))

print('ЭЦП RSA:')
print('КОРОТКИЙ ТЕКСТ:')
rsacipher('31', '7', input_for_cipher_short())
print('ДЛИННЫЙ ТЕКСТ:')
rsacipher('31', '7', input_for_cipher_long())

```

Тестирование:

```

/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab09_24_rsa.py
ЭЦП RSA:
КОРОТКИЙ ТЕКСТ:
p: True
q: True
N = 217
El = 180

```

```
Е = 7
1 Е подходит
D = 103
Открытый ключ e=7 n=217
Секретный ключ d=103 n=217
Длина исходного сообщения 39 символов
Хэш сообщения 121
Значение подписи: 100
Значение проверки хэша = 121

ДЛИННЫЙ ТЕКСТ:
p: True
q: True
N = 217
E1 = 180
Е = 7
1 Е подходит
D = 103
Открытый ключ e=7 n=217
Секретный ключ d=103 n=217
Длина исходного сообщения 1087 символов
Хэш сообщения 144
Значение подписи: 165
Значение проверки хэша = 144
```


25.El Gamal

Для того чтобы генерировать пару ключей (открытый ключ - секретный ключ), сначала выбирают некоторое большое простое число P и большое целое число G , причем $G < P$. Отправитель и получатель подписанного документа используют при вычислениях близкие большие целые числа P (~10308 или ~21024) и G (~10154 или ~2512), которые не являются секретными.

Код программы:

```
from math import gcd
import random
from base import alphabet, input_for_cipher_short, input_for_cipher_long,
output_from_decrypted
alphavit = {'a': 0, 'б': 1, 'в': 2, 'г': 3, 'д': 4,
            'е': 5, 'ё': 6, 'ж': 7, 'з': 8, 'и': 9, 'й': 10,
            'к': 11, 'л': 12, 'м': 13, 'н': 14, 'о': 15,
            'п': 16, 'р': 17, 'с': 18, 'т': 19, 'у': 20,
            'ф': 21, 'х': 22, 'ц': 23, 'ч': 24, 'ш': 25,
            'щ': 26, 'ъ': 27, 'ы': 28, 'ь': 29, 'э': 30,
            'ю': 31, 'я': 32
            }

def IsPrime(n):
    d = 2
    while n % d != 0:
        d += 1
    return d == n

def modInverse(e, el):
    e = e % el
    for x in range(1, el):
        if ((e * x) % el == 1):
            return x
    return 1

def is_prime(num, test_count):
    if num == 1:
        return False
    if test_count >= num:
        test_count = num - 1
    for x in range(test_count):
        val = random.randint(1, num - 1)
        if pow(val, num-1, num) != 1:
            return False
    return True

def gen_prime(n):
    found_prime = False
    while not found_prime:
        p = random.randint(2**(n-1), 2**n)
        if is_prime(p, 1000):
            return p
```

```

def hash_value(mod, alpha_code_msg):
    i = 0
    hashing_value = 1
    while i < len(alpha_code_msg):
        hashing_value = (((hashing_value-1) + int(alpha_code_msg[i]))**2) %
mod
        i += 1
    return hashing_value

def egccipher(clearText):
    p = gen_prime(10)
    print("P =", p)
    g = random.randint(2, p-1)
    print("G =", g)

    x = random.randint(2, p-2)
    y = (g**x) % p
    print("Открытый ключ (Y)={}, Секретный ключ (X)={}".format(y, x))

    msg = clearText
    msg_list = list(msg)
    alpha_code_msg = list()
    for i in range(len(msg_list)):
        alpha_code_msg.append(int(alphavit.get(msg_list[i])))
    print("Длина исходного сообщения {} символов".format(len(alpha_code_msg)))

    hash_code_msg = hash_value(p, alpha_code_msg)
    print("Хэш сообщения:= {}".format(hash_code_msg))

    k = 1
    while True:
        k = random.randint(1, p-2)
        if gcd(k, p-1) == 1:
            print("K =", k)
            break

    a = (g**k) % p

    b = (hash_code_msg - (x*a)) % (p-1)
    print("Значение подписи:S={},{}".format(a, b))
    b = modInverse(k, p-1) * ((hash_code_msg - (x * a)) % (p-1))

    check_hash_value = hash_value(p, alpha_code_msg)
    a_1 = ((y**a) * (a**b)) % p
    print("A1={}".format(a_1))
    a_2 = (g**check_hash_value) % p
    print("A2={}".format(a_2))
    if a_1 == a_2:
        print("Подпись верна\n")
    else:
        print("Подпись неверна")

```

```
print('ЭЦП Elgamal:')
print('КОРОТКИЙ ТЕКСТ:')
egcipher(input_for_cipher_short())
print('ДЛИННЫЙ ТЕКСТ:')
egcipher(input_for_cipher_long())
```

Тестирование:

```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab09_25_elgamal.py
```

```
ЭЦП Elgamal:
КОРОТКИЙ ТЕКСТ:
P = 907
G = 875
Открытый ключ (Y)=665, Секретный ключ (X)=617
Длина исходного сообщения 39 символов
Хэш сообщения:= 376
K = 769
Значение подписи:S=550,776
A1=194
A2=194
Подпись верна

ДЛИННЫЙ ТЕКСТ:
P = 587
G = 410
Открытый ключ (Y)=109, Секретный ключ (X)=161
Длина исходного сообщения 1087 символов
Хэш сообщения:= 423
K = 35
Значение подписи:S=226,369
A1=102
A2=102
Подпись верна
```

Блок J: СТАНДАРТЫ ЦИФРОВЫХ ПОДПИСЕЙ

26.ГОСТ Р 34.10-94

p - большое простое число длиной от 509 до 512 бит либо от 1020 до 1024 бит;

q - простой сомножитель числа $(p - 1)$, имеющий длину 254...256 бит;

a - любое число, большее 1 и меньшее $(p-1)$, причем такое, что $aq \bmod p = 1$;

x - некоторое число, меньшее q ;

$y = ax \bmod p$.

Кроме того, этот алгоритм использует однонаправленную хэш-функцию $H(x)$. Стандарт ГОСТ Р 34.11-94 определяет хэш-функцию, основанную на использовании стандартного симметричного алгоритма ГОСТ 28147-89.

Код программы:

```
from base import alphabet, input_for_cipher_short, input_for_cipher_long,
output_from_decrypted

alphavit = {'a': 0, 'б': 1, 'в': 2, 'г': 3, 'д': 4,
            'е': 5, 'ё': 6, 'ж': 7, 'з': 8, 'и': 9, 'й': 10,
            'к': 11, 'л': 12, 'м': 13, 'н': 14, 'о': 15,
            'п': 16, 'р': 17, 'с': 18, 'т': 19, 'у': 20,
            'ф': 21, 'х': 22, 'ц': 23, 'ч': 24, 'ш': 25,
            'щ': 26, 'ъ': 27, 'ы': 28, 'ь': 29, 'э': 30,
            'ю': 31, 'я': 32
            }

def ciphergostd(clearText):
    array = []
    flag = False
    for s in range(50, 1000):
        for i in range(2, s):
            if s % i == 0:
                flag = True
                break
        if flag == False:
            array.append(s)
            flag = False
    p = 31
    print("p = ", p)
    q = 5
    print("q = ", q)
    a = 2
    print("a =", a)

    array2 = []
    flag2 = False
    for s in range(2, q):
        for i in range(2, s):
```

```

        if s % i == 0:
            flag2 = True
            break
    if flag2 == False:
        array2.append(s)
    flag2 = False

```

```

x = 3
print("x = ", x)
y = a**x % p
k = 4
print("k = ", k)
r = (a**k % p) % q

```

```

msg = clearText
msg_list = list(msg)
alpha_code_msg = list()
for i in range(len(msg_list)):
    alpha_code_msg.append(int(alphavit.get(msg_list[i])))
print("Длина исходного сообщения {} символов".format(len(alpha_code_msg)))
hash_code_msg = hash_value(p, alpha_code_msg)
print("Хэш сообщения:= {}".format(hash_code_msg))

```

```

s = (x*r+k*hash_code_msg) % q

```

```

print("Цифровая подпись = ", r % (2**256), ",", s % (2**256))

```

```

v = (hash_code_msg**(q-2)) % q
z1 = s*v % q
z2 = ((q-r)*v) % q
u = (((a**z1)*(y**z2)) % p) % q
print(r, " = ", u)
if u == r:
    print("r = u, следовательно:")
    print("Подпись верна\n")
else:
    print("Подпись неверна")

```

```

def hash_value(n, alpha_code):
    i = 0
    hash = 1
    while i < len(alpha_code):
        hash = (((hash-1) + int(alpha_code[i]))**2) % n
        i += 1
    return hash

```

```

print('ГОСТ Р 34.10-94:')
print('КОРОТКИЙ ТЕКСТ:')
ciphergostd(input_for_cipher_short())
print('ДЛИННЫЙ ТЕКСТ:')
ciphergostd(input_for_cipher_long())

```

Тестирование:

```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab10_26_gost94.py
```

ГОСТ Р 34.10-94:

КОРОТКИЙ ТЕКСТ:

p = 31

q = 5

a = 2

x = 3

k = 4

Длина исходного сообщения 39 символов

Хэш сообщения:= 28

Цифровая подпись = 1 , 0

1 = 1

r = u, следовательно:

Подпись верна

ДЛИННЫЙ ТЕКСТ:

p = 31

q = 5

a = 2

x = 3

k = 4

Длина исходного сообщения 1087 символов

Хэш сообщения:= 20

Цифровая подпись = 1 , 3

1 = 1

r = u, следовательно:

Подпись верна

27.ГОСТ Р 34.10-2012

Для сообщества пользователей выбирается общая эллиптическая кривая $E_p(a, b)$ и точка G на ней, такая, что $G, [2]G, [3]G, \dots, [q]G$ суть различные точки, и $[q]G = O$ для некоторого простого числа q (длина числа q равна 256 бит). Каждый пользователь U выбирает случайное число x_u (секретный ключ), $0 < x_u < q$, и вычисляет точку на кривой $Y_u = [x_u]G$ (открытый ключ). Параметры кривой и список открытых ключей передаются всем пользователям.

Чтобы подписать сообщение, пользователь A делает следующее:

1. Вычисляет значение хеш-функции сообщения $h = h()$;
2. Выбирает случайно число k , $0 < k < q$;
3. Вычисляет $P = [k]G = (x, y)$;
4. Вычисляет $r = x \bmod q$ (при $r = 0$ возвращается к шагу 2);
5. Вычисляет $s = (kh + r x_a) \bmod q$ (при $s = 0$ возвращается к шагу 2);
6. Подписывает сообщение парой чисел (r, s) .

Для проверки подписанного сообщения $(; r, s)$ любой пользователь, знающий открытый ключ Y_A , делает следующее:

1. Вычисляет $h = h()$;
2. Убеждается, что $0 < r, s < q$;
3. Вычисляет $u_1 = s \cdot h^{-1} \bmod q$ и $u_2 = -r \cdot h^{-1} \bmod q$;
4. Вычисляет композицию точек на кривой $P = [u_1]G + [u_2]Y_A = (x, y)$ и, если $P = O$, отвергает подпись;
5. Если $x \bmod q = r$, принимает подпись, в противном случае отвергает ее.

Код программы:

```
import random
import collections
from base import alphabet, input_for_cipher_short, input_for_cipher_long,
output_from_decrypted

alphabet_lower = {'a': 0, 'б': 1, 'в': 2, 'г': 3, 'д': 4,
                  'е': 5, 'ё': 6, 'ж': 7, 'з': 8, 'и': 9, 'й': 10,
                  'к': 11, 'л': 12, 'м': 13, 'н': 14, 'о': 15,
                  'п': 16, 'р': 17, 'с': 18, 'т': 19, 'у': 20,
                  'ф': 21, 'х': 22, 'ц': 23, 'ч': 24, 'ш': 25,
                  'щ': 26, 'ъ': 27, 'ы': 28, 'ь': 29, 'э': 30,
                  'ю': 31, 'я': 32
                  }

class Point:
    def __init__(self, x_init, y_init):
        self.x = x_init
        self.y = y_init

    def shift(self, x, y):
        self.x += x
        self.y += y
```

```

def __repr__(self):
    return "".join(["( x=", str(self.x), ", y=", str(self.y), ")"])

x_1 = 0
y_1 = 0

EllipticCurve = collections.namedtuple(
    'EllipticCurve', 'name p q_mod a b q g n h')
curve = EllipticCurve(
    'secp256k1',
    p=0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffc2f,
    q_mod=0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffc2f,

    a=7,
    b=11,

    g=(0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798,
        0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8),
    q=(0xA0434D9E47F3C86235477C7B1AE6AE5D3442D49B1943C2B752A68E2A47E247C7,
        0x893ABA425419BC27A3B6C7E693A24C696F794C2ED877A1593CBEE53B037368D7),

    n=0xfffffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141,

    h=1,
)

def ciphergosto(clearText):
    msg = clearText
    msg_list = list(msg)
    alpha_code_msg = list()
    for i in range(len(msg_list)):
        alpha_code_msg.append(int(alphabet_lower.get(msg_list[i])))
    print("Длина исходного сообщения {} символов".format(len(alpha_code_msg)))
    print("Q mod", int(curve.q_mod))
    print("P mod", int(curve.p))

    hash_code_msg = hash_value(curve.p, alpha_code_msg)
    print("Хэш сообщения:{}".format(hash_code_msg))

    e = hash_code_msg % curve.q_mod
    print("E={}".format(e))

    k = random.randint(1, curve.q_mod)
    print("K={}".format(k))

    d = 10
    print("D={}".format(d))
    x, y = scalar_mult(k, curve.g)
    point_c = Point(x, y)
    print("Point_C={}".format(point_c))
    r = point_c.x % curve.q_mod
    print("R={}".format(r))

```



```

s = (r*curve.p + k*e) % curve.q_mod
print("S={}".format(s))

v = inverse_mod(e, curve.p)
print("V={}".format(v))
z1 = (s*v) % curve.q_mod
z2 = ((curve.p-r)*v) % curve.q_mod
x_1, y_1 = scalar_mult(d, curve.g)
print("Point_Q=( x={}, y={} )".format(x_1, y_1))
point_c_new = Point(x, y)
x, y = point_add(scalar_mult(z1, curve.g),
                  scalar_mult(z2, curve.g))
r_1 = point_c_new.x % curve.q_mod
print("R_new={}".format(r_1))
if r == r_1:
    print("Подпись прошла проверку!\n")
else:
    print("Ошибка проверки!")

def hash_value(mod, alpha_code_msg):
    i = 0
    hashing_value = 1
    while i < len(alpha_code_msg):
        hashing_value = (
            (hashing_value-1) + int(alpha_code_msg[i]))**2) % curve.p
        i += 1
    return hashing_value

def is_on_curve(point):
    if point is None:
        return True
    x, y = point
    return (y * y - x * x * x - curve.a * x - curve.b) % curve.p == 0

def point_neg(point):
    if point is None:
        return None
    x, y = point
    result = (x, -y % curve.p)
    return result

def inverse_mod(k, p):
    if k == 0:
        raise ZeroDivisionError('деление на 0')
    if k < 0:
        return p - inverse_mod(-k, p)

    s, old_s = 0, 1
    t, old_t = 1, 0
    r, old_r = p, k

    while r != 0:
        quotient = old_r // r

```

```

old_r, r = r, old_r - quotient * r
old_s, s = s, old_s - quotient * s
old_t, t = t, old_t - quotient * t

```

```

gcd, x, y = old_r, old_s, old_t

```

```

assert gcd == 1
assert (k * x) % p == 1

```

```

return x % p

```

```

def point_add(point1, point2):
    if point1 is None:
        return point2
    if point2 is None:
        return point1
    x1, y1 = point1
    x2, y2 = point2

    if x1 == x2 and y1 != y2:
        return None
    if x1 == x2:
        m = (3 * x1 * x1 + curve.a) * inverse_mod(2 * y1, curve.p)
    else:
        m = (y1 - y2) * inverse_mod(x1 - x2, curve.p)

    x3 = m * m - x1 - x2
    y3 = y1 + m * (x3 - x1)
    result = (x3 % curve.p,
              -y3 % curve.p)
    return result

```

```

def scalar_mult(k, point):
    if k % curve.n == 0 or point is None:
        return None
    if k < 0:
        return scalar_mult(-k, point_neg(point))

    result = None
    addend = point

    while k:
        if k & 1:
            result = point_add(result, addend)
        addend = point_add(addend, addend)
        k >>= 1
    return result

```

```

print('ГОСТ Р 34.10-2012:')
print('КОРОТКИЙ ТЕКСТ:')
ciphergosto(input_for_cipher_short())
print('ДЛИННЫЙ ТЕКСТ:')
ciphergosto(input_for_cipher_long())

```

Тестирование:

```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab10_27_gost2012.py

ГОСТ Р 34.10-2012:
КОРОТКИЙ ТЕКСТ:
Длина исходного сообщения 39 символов
Q mod 115792089237210883131902140479076077470404524942491262870694982560773809634351
P mod 115792089237316195423570985008687907853269984665640564039457584007908834671663
Хэш
сообщения:=10046560596039240554532661598912076547906755950821571527490860166833943231
2127
E=100465605960392405545326615989120765479067559508215715274908601668339432312127
K=43019610968597533575694927402782043341666815463049898570394327923006212119082
D=10
Point_C=(
x=80553680982383746563554500880210053620115828941187278030733140405035128490425,
y=74824895621507411931400280064663892593331393722726527632339907349142076474785)
R=80553680982383746563554500880210053620115828941187278030733140405035128490425
S=35581294126978857713435754913480675793101545360322986846601181291383678561299
V=20529166449300022691683847027495261733238752076834297463717225331510851960291
Point_Q=(
x=109805586211166206629432866892583231117554510260596600142888290125507993067118,
y=51243083235504058321191534323736250822297443681753984114121156474938550647252 )
R_new=80553680982383746563554500880210053620115828941187278030733140405035128490425
Подпись прошла проверку!

ДЛИННЫЙ ТЕКСТ:
Длина исходного сообщения 1087 символов
Q mod 115792089237210883131902140479076077470404524942491262870694982560773809634351
P mod 115792089237316195423570985008687907853269984665640564039457584007908834671663
Хэш
сообщения:=44642821119098115386834658767856322616357022834257596733930388099678580390
171
E=44642821119098115386834658767856322616357022834257596733930388099678580390171
K=48431864128040811700370753657562279379539026068468713299770002081598293602354
D=10
Point_C=(
x=44800459081610348166673508108385128878705452232210702982677761178660890566077,
y=13195091569128410442209436744524910967401329637352914815201625579914336769926)
R=44800459081610348166673508108385128878705452232210702982677761178660890566077
S=110804626836197049152945897279988234683557906819154210898078501121592663046804
V=24078682445532384626955465808779465306725321854082666482660247247597808478951
Point_Q=(
x=109805586211166206629432866892583231117554510260596600142888290125507993067118,
y=51243083235504058321191534323736250822297443681753984114121156474938550647252 )
R_new=44800459081610348166673508108385128878705452232210702982677761178660890566077
Подпись прошла проверку!
```

28. ОБМЕН КЛЮЧАМИ ПО ДИФФИ-ХЕЛЛМАНУ

В протоколе обмена секретными ключами предполагается, что все пользователи знают некоторые числа n и a ($1 < a < n$). Для выработки общего секретного ключа пользователи A и B должны проделать следующую процедуру:

1. Определить секретные ключи пользователей K_A и K_B .
2. Для этого каждый пользователь независимо выбирает случайные числа из интервала $(2, \dots, n-1)$.
3. Вычислить открытые ключи пользователей Y_A и Y_B : $Y = a^K \bmod n$
4. Обменяться ключами Y_A и Y_B по открытому каналу связи.
5. Независимо определить общий секретный ключ K : $K_A = Y_B^{K_A} \bmod n$ $K_B = Y_A^{K_B} \bmod n$.

$$K_A = K_B = K$$

Код программы:

```
a = int(input("Введите число a: "))
n = int(input("Введите число n, n должно быть больше a: "))
ka = int(input("Введите число ka: "))
Ya = a**ka % n
print ("Ваш Ya = ", Ya)
Yb = int(input("Введите число Yb, которое прислал собеседник: "))
K = (a**(Ya*Yb))%n
print ("Ваш общий ключ: ", K)
```

Тестирование:

```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab11_28_dh.py
Введите число a: 10
Введите число n, n должно быть больше a: 30
Введите число ka: 4
Ваш Ya = 10
Введите число Yb, которое прислал собеседник: 20
Ваш общий ключ: 10
```