МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

**ЛАБОРАТОРНЫЕ РАБОТЫ ПО ПРЕДМЕТУ**

«Программирование криптографических алгоритмов»

**Выполнил:**

Барышников С.С. гр. 191-351

**Преподаватель:**

Бутакова Н.Г.

Москва 2021 г.

**Содержание**

[Аннотация 3](#_Toc89341693)

[Постоянный модуль 4](#_Toc89341694)

[Блок А: ШИФРЫ ОДНОЗНАЧНОЙ ЗАМЕНЫ 5](#_Toc89341695)

[1. Шифр простой замены АТБАШ 5](#_Toc89341696)

[2. ШИФР ЦЕЗАРЯ 7](#_Toc89341697)

[3. Квадрат Полибия 9](#_Toc89341698)

[Блок В: ШИФРЫ МНОГОЗНАЧНОЙ ЗАМЕНЫ 12](#_Toc89341699)

[4. Шифр Тритемия 12](#_Toc89341700)

[5. Шифр Белазо 15](#_Toc89341701)

[Блок С: ШИФРЫ БЛОЧНОЙ ЗАМЕНЫ 18](#_Toc89341702)

[8. Матричный шифр 18](#_Toc89341703)

[9. Шифр Плейфера 21](#_Toc89341704)

[D: ШИФРЫ ПЕРЕСТАНОВКИ 26](#_Toc89341705)

[10. Шифр вертикальной перестановки 26](#_Toc89341706)

[11. Решетка Кардано 29](#_Toc89341707)

[E: ШИФРЫ ГАММИРОВАНИЯ 33](#_Toc89341708)

[13. Одноразовый блокнот К.Шеннона 33](#_Toc89341716)

[14. Гаммирование ГОСТ 28147-89 38](#_Toc89341717)

[F: ПОТОЧНЫЕ ШИФРЫ 43](#_Toc89341718)

[15. А5 /1 43](#_Toc89341719)

# 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 стопятьдесят или двести слов средней величины. Но, если злоупотреблять предлогами, союзами и другими частями речи на один или два символа, то количество слов неизменно возрастает. В копирайтерской деятельности принято считать тысячи с пробелами или без. Учет пробелов увеличивает объем текста примерно на сто или двести символов именно столько раз мы разделяем слова свободным пространством. Считать пробелы заказчики не любят, так как это пустое место. Однако некоторые фирмы и биржи видят справедливым ставить стоимость за тысячу символов с пробелами, считая последние важным элементом качественного восприятия. Согласитесь, читать слитный текст без единого пропуска, никто не будет. Но большинству нужна цена за тысячу знаков без пробелов.

**Интерфейс:** #в разработке#

# Постоянный модуль

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

import re

alphabet = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя"

dict = {'.': 'тчк', ',': 'зпт'}

def replace\_all\_to(input\_text, dict):

    input\_text = input\_text.replace(' ', '')

    for i, j in dict.items():

        input\_text = input\_text.replace(i, j)

    return input\_text

def replace\_all\_from(input\_text, dict):

    for i, j in dict.items():

        input\_text = input\_text.replace(j, i)

    return input\_text

def file\_to\_string(name):

    with open(name) as f:

        input\_short\_text = " ".join([l.rstrip() for l in f]) + ' '

    return input\_short\_text.lower()

def input\_for\_cipher\_short():

    return replace\_all\_to(file\_to\_string('short.txt'), dict)

def input\_for\_cipher\_long():

    return replace\_all\_to(file\_to\_string('long.txt'), dict)

def output\_from\_decrypted(decrypted\_text):

    return replace\_all\_from(decrypted\_text, dict)

# Блок А: ШИФРЫ ОДНОЗНАЧНОЙ ЗАМЕНЫ

## 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

# функция преобразования текста

def atbash(input):

    return input.translate(str.maketrans(

        alphabet + alphabet.upper(), alphabet[::-1] + alphabet.upper()[::-1]))

# вывод результатов работы программы

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:**

root@DESKTOP-05UI9FD:~/crypt# /bin/python3 /root/crypt/lab01\_1\_atbash.py

ШИФР АТБАШ:

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

эоъта чпм поцуцэд ц рмуцэд съ шылм зъурэъфя мзф

Расшифрованный текст:

время, приливы и отливы не ждут человека.

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

эрм поцтъо нмямгц ся мдназл нцтэрурэ мзф вмр ырнмямрзср тяуъсгфцх мъфнм чпм рпмцтяугср прыйрыаёцх ыуа фяомрзъф мрэяорэ э цсмъосъм цуц тяьячцсяй цуц ыуа съюругжцй цскротяицрссдй плюуцфяицх мзф э мяфрт мъфнмъ оъыфр юдэяъм юруъъ ыэлй цуц мощй яючяиъэ ц рюдзср рыцс прычяьрурэрф мзф ср тршср ц юъч съьр мзф ся мдназл нцтэрурэ оъфртъсырэяср цнпругчрэямг рыцс цуц ыэя фубзя ц рысл фяомцсл мзф мъфнм ся мдназл нцтэрурэ вмр нфругфр поцтъоср нурэ эпо нмямцнмцфя прфячдэяъм чпм змр мдназя эфубзяъм э нъюа нмрпамгыънам цуц ыэънмц нурэ ноъысъх эъуцзцсд мзф ср чпм ънуц чурлпрмоъюуамг поъыурьятц чпм нрбчятц ц ыольцтц зянматц оъзц ся рыцс цуц ыэя нцтэруя чпм мр фруцзънмэр нурэ съцчтъсср эрчоянмяъм мзф э фрпцояхмъонфрх ыъамъугсрнмц поцсамр нзцмямг мдназц н порюъуятц цуц юъч мзф лзъм порюъурэ лэъуцзцэяъм рюеът мъфнмя поцтъоср ся нмр цуц ыэънмц нцтэрурэ цтъсср нмругфр ояч тд оячыъуаът нурэя нэрюрысдт порнмояснмэрт мзф нзцмямг порюъуд чяфячзцфц съ убюам чпм мяф фяф вмр плнмръ тънмр мзф рысяфр съфрмродъ кцотд ц юцошц эцыам нпояэъыуцэдт нмяэцмг нмрцтрнмг чя мдназл нцтэрурэ н порюъуятц чпм нзцмяа прнуъысцъ эяшсдт вуътъсмрт фязънмэъссрьр эрнпоцамца мзф нрьуянцмънг чпм зцмямг нуцмсдх мъфнм юъч ъыцсрьр порплнфя чпм сцфмр съ юлыъм мзф ср юругжцснмэл слшся иъся чя мдназл чсяфрэ юъч порюъурэ мзф

Расшифрованный текст:

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 стопятьдесят 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.

## 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 = int(input('Введите ключ: '))

# функция шифровки

def caesar\_encode(input, step):

    return input.translate(

        str.maketrans(alphabet, alphabet[step:] + alphabet[:step]))

# функция дешифровки

def caesar\_decode(input, step):

    return input.translate(

        str.maketrans(alphabet[step:] + alphabet[:step], alphabet))

# вывод результатов работы программы

print(f'''

ШИФР ЦЕЗАРЯ:

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:**

root@DESKTOP-05UI9FD:~/crypt# /bin/python3 /root/crypt/lab01\_2\_caesar.py

ШИФР ЦЕЗАРЯ:

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

бпдлю жос опзкзбъ з нскзбъ мд ёгтс цдкнбдйя сцй

Расшифрованный текст:

время, приливы и отливы не ждут человека.

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

бнс опзлдп рсясыз мя сърюцт рзлбнкнб сцй ьсн гнрсяснцмн лякдмыйзи сдйрс жос носзлякымн онгфнгюшзи гкю йяпснцдй снбяпнб б змсдпмдс зкз лявяжзмяф зкз гкю мданкычзф змунпляхзнммъф отакзйяхзи сцй б сяйнл сдйрсд пдгйн аъбядс анкдд гбтф зкз спеф яажяхдб з наъцмн нгзм онгжявнкнбнй сцй мн лнёмн з адж мдвн сцй мя сърюцт рзлбнкнб пдйнлдмгнбямн зронкыжнбясы нгзм зкз гбя йкэця з нгмт йяпсзмт сцй сдйрс мя сърюцт рзлбнкнб ьсн рйнкыйн опзлдпмн ркнб боп рсясзрсзйя онйяжъбядс жос цсн сърюця бйкэцядс б рдаю рсн оюсыгдрюс зкз гбдрсз ркнб рпдгмди бдкзцзмъ сцй мн жос дркз жкнтонспдакюсы опдгкнвялз жос рнэжялз з гптвзлз цярсюлз пдцз мя нгзм зкз гбя рзлбнкя жос сн йнкзцдрсбн ркнб мдзжлдммн бнжпярсядс сцй б йнозпяисдпрйни гдюсдкымнрсз опзмюсн рцзсясы сърюцз р опнадкялз зкз адж сцй тцдс опнадкнб тбдкзцзбядс нащдл сдйрся опзлдпмн мя рсн зкз гбдрсз рзлбнкнб злдммн рснкыйн пяж лъ пяжгдкюдл ркнбя рбнангмъл опнрспямрсбнл сцй рцзсясы опнадкъ жяйяжцзйз мд кэаюс жос сяй йяй ьсн отрснд лдрсн сцй нгмяйн мдйнснпъд узплъ з азпёз бзгюс ропябдгкзбъл рсябзсы рснзлнрсы жя сърюцт рзлбнкнб р опнадкялз жос рцзсяю онркдгмзд бяёмъл ькдлдмснл йяцдрсбдммнвн бнропзюсзю сцй рнвкярзсдры жос цзсясы ркзсмъи сдйрс адж дгзмнвн опнотрйя жос мзйсн мд атгдс сцй мн анкычзмрсбт мтёмя хдмя жя сърюцт жмяйнб адж опнадкнб сцй

Расшифрованный текст:

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.

## 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

# квадрат полибия

hard\_dictionary = {"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"}

# функция шифровки

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

# функция дешифровки

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

# вывод результатов работы программы

print(f'''

КВАДРАТ ПОЛИБИЯ:

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:**

root@DESKTOP-05UI9FD:~/crypt# /bin/python3 /root/crypt/lab01\_3\_square.py

КВАДРАТ ПОЛИБИЯ:

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

1336163263 233542 35362431241355 24 344231241355 3316 22154342 5116313413162611 425126

Расшифрованный текст:

время, приливы и отливы не ждут человека.

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

133442 353624321636 414211425624 3311 425541635143 4124321334313413 425126 614234 15344142114234513334 321131163356262425 4216264142 233542 34354224321131563334 35341545341563532425 153163 2611364234511626 42341311363413 13 2433421636331642 243124 321114112324331145 243124 153163 331612343156522445 2433443436321146243433335545 35431231242611462425 425126 13 4211263432 421626414216 3616152634 125513111642 1234311616 15134345 243124 42362145 11122311461613 24 341255513334 34152433 353415231114343134133426 425126 3334 3234223334 24 121623 33161434 425126 3311 425541635143 4124321334313413 36162634321633153413113334 244135343156233413114256 34152433 243124 151311 2631625111 24 34153343 26113642243343 425126 4216264142 3311 425541635143 4124321334313413 614234 41263431562634 3536243216363334 41313413 133536 41421142244142242611 35342611235513111642 233542 514234 425541635111 1326316251111642 13 41161263 414234 356342561516416342 243124 151316414224 41313413 41361615331625 1316312451243355 425126 3334 233542 16413124 2331344335344236161231634256 35361615313414113224 233542 41346223113224 24 15364314243224 51114142633224 36165124 3311 34152433 243124 151311 41243213343111 233542 4234 26343124511641421334 41313413 331624233216333334 13342336114142111642 425126 13 2634352436112542163641263425 151663421631563334414224 35362433634234 41512442114256 425541635124 41 353634121631113224 243124 121623 425126 43511642 3536341216313413 4313163124512413111642 3412541632 421626414211 3536243216363334 3311 414234 243124 151316414224 4124321334313413 243216333334 41423431562634 361123 3255 361123151631631632 4131341311 411334123415335532 35363441423611334142133432 425126 41512442114256 35363412163155 231126112351242624 3316 3162126342 233542 421126 261126 614234 354341423416 3216414234 425126 341533112634 331626344234365516 4424363255 24 1224362224 1324156342 413536111316153124135532 41421113244256 414234243234414256 2311 425541635143 4124321334313413 41 353634121631113224 233542 415124421163 353441311615332416 131122335532 613116321633423432 26115116414213163333341434 13344135362463422463 425126 4134143111412442164156 233542 512442114256 41312442335525 4216264142 121623 16152433341434 3536343543412611 233542 3324264234 3316 1243151642 425126 3334 1234315652243341421343 3343223311 46163311 2311 425541635143 233311263413 121623 3536341216313413 425126

Расшифрованный текст:

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.

# Блок В: ШИФРЫ МНОГОЗНАЧНОЙ ЗАМЕНЫ

## 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

# функция шифровки

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

# функция дешифровки

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

# вывод результатов работы программы

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:

всжпгмхщчщтцфоичюгэыхпгыюьммтбимбелвхып

Расшифрованный текст:

время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

впфтфнтлшъькюицьпгмдтлизаеыижкярцкюфсзучщзышвщыъоыхяоюэяиьгкмглпмотйогпбухччнаърмзшъхютяхжйжряёолаярпдемтшлшцоэфшцыпусъьвхладвюжыкгаомюымъофьчъчлгцээюмзгзцвагщрдёпхйвувнтсшлтъьпьссщюсмфущзчёдюяяюрузлйфуйъёзпиапнхьпкзябвшюджжэвыялйнвпмхыпухфчршъхоучюцхвжмбешлхмыфсриндспузчмушчръсэсрябьёеегфбиэьпъндйплпнйизухигмцэуюеэуезяеллсёовиртовхяцеюыътчныщэсндбеядвугзйлейгпнпуотжешъиьуэяцщпааэуьршчэлвкофрнтьувыезсужбкряпафсрдёгехйфэямыпжкиедзхчошучлььссьфъучяэмяеулсёйлёотёуоммсхышшэьсоъогнвдщщвёыщжддмррояйгрнокшьмушхеобгряьеънаёщияекжкиедгхнтицфйтыяаэъяэыкщжищжкёйюкгззнжрсузхпщйьюмтбвзьфюгеязшгмамоинйежвцйсгхчыэфьючэпдбелюичкхмцньхзсртсейсжфстцфнтцвзьёкщзёзжъяжкясбемкъмжёлъчкерщауъвдтгеюгьижииьорнимкжёйчщрътчныщэсщючбвдзйжецёкнюжмьтгликтжнцьчыпобтаувшгсдзйимонюмсудсрчэсщатэляйюаятчяпэцвшбсджёлдлягвцмхщъифхлчкбюаедёгъгьйлинузмгнбмссйрхъчъёрбцяьлъуьеьящбэщннмоявёёомжбущыймфяйавяъявёзтяшлхмыфсриндспузшчщшлршнышшбеелюиччиимзввмиёдгктбуефочръаэьысйшёзшьёжиюкямрпсквцнещяуыщошнашгцдпьеиоблъншойзтоэмцйршъйотррцьюуавдгейигщвкокжйтппзешйлъыбхщыоэымйыбёзшкёмбиьугмаивхяцеюырчкцыппфшбгвхъвёълсё

Расшифрованный текст:

вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходящийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпубликаций.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.номожноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднукартину.текстнатысячусимволовэтосколькопримернословястатистикапоказывает,чтотысячавключаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблятьпредлогами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеизменновозрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиилибез.учетпробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименностолькоразмыразделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,таккакэтопустоеместо.однаконекоторыефирмыибирживидятсправедливымставитьстоимостьзатысячусимволовспробелами,считаяпоследниеважнымэлементомкачественноговосприятия.согласитесь,читатьслитныйтекстбезединогопропуска,никтонебудет.нобольшинствунужнаценазатысячузнаковбезпробелов.

## Шифр Белазо

В 1553 Джованни Баттиста Белазо предложил использовать для многоалфавитного шифра буквенный, легко запо­минаемый 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 исполь­зовался 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 = str(input('Введите ключ: '))

# функция шифровки

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

# функция дешифровки

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

# вывод результатов работы программы

print(f'''

Шифр Тритемия:

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

Введите ключ: ключик

Шифр Тритемия:

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

мьгдзтъюнзсцунщачэцфатцпспсйапцъаьукэги

Расшифрованный текст:

время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

мържщучроиыкэзжеиэёээоььушаёфщмюхвёэщпмиыкэъхеччкчгеехухрьуьэунйчъэфкчфжшънёмащпэрсфочэвиыэъхьуэщнюзчммфлйнышррафучлбчрушлуафуочэенлщчъпсаущтёщчквжёцшёбнкйцуцюнсфэгищыкхъкйнхьюгзнохъятккпюяёфпппакюуцфрзоакмёчяпмфмшдвшъмысшъъвяинщчмщчхэгиеччщтлёслпульлщэгиеиэёээоььушаёфщмьгвччпщвёккшъжишщцзёёккэзмысшучжыккхчьоиущплкукыюжеьэвцрьуьэщюйдьйгсисчмъйёкзэъпвчцжцмжщучроечьцъа?иыкэфпйсхкымвитёнюьытъюхйчэёээоимхчьоипэнпьййьюмжзэжпгизэучжыкпьюжифщмэоьмшпхаьфувфлтывхщмяшэпэйарцщянёыыпмйцыжъьгыфщнлкаръээмхркчфжыщюнфкаакьюэдсыпгжеищофлафуонюисчмъйчръэюмвчцуггиымщэйёкшпфёдншшъаёрыкэрчнээгищущъфочтэпьпвчфорэйнцжщмиыуъьжезэщэхаыкэзртъйвфпжщщлрйчхуучжшнтэгикапэыоёйпцъаккпцфхаккпюмшгпчюгвъэкыоахпыщмеиьэъжгсомрпйсьушаёфщмфкьцшщэрёфжхъочрчёьюямпцкгдъцщнюикщлъведчъьмиыыкщпйкщчюхвъвуююйеъыъяьфётличрвуцженцимэйръэюювукхирёшюьюмьхпьюмйахщплчущшриёыщыжглсычжжшсысфаамйээнзимппйакёчэрчкуэзпйчучъпйеткющизвюэждкщцъаишыщмггичуунйъвуююцшщьчгыцупнююцёчийьхпшюмдукврпйкпшщмъчмщэнзсйэфэйахьъбгиьуюгиетъюхаыкэзпгсэшжзйнхьюяьрпофлёлщъьмжььхлёжышуцрёцплявьыэвцлёйщцзцацьэнсеьсшлфьцктлртъйвяёеихщняьръыъяьфщмюхв

Расшифрованный текст:

вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходящийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпубликаций.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.номожноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднукартину.текстнатысячусимволовэтосколькопримернослов?статистикапоказывает,чтотысячавключаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблятьпредлогами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеизменновозрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиилибез.учетпробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименностолькоразмыразделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,таккакэтопустоеместо.однаконекоторыефирмыибирживидятсправедливымставитьстоимостьзатысячусимволовспробелами,считаяпоследниеважнымэлементомкачественноговосприятия.согласитесь,читатьслитныйтекстбезединогопропуска,никтонебудет.нобольшинствунужнаценазатысячузнаковбезпробелов.

# Блок С: ШИФРЫ БЛОЧНОЙ ЗАМЕНЫ

## 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

inp = input('Введите матрицу в строку через пробел: ')

inp = inp.split(' ')

# 3 10 20 20 19 17 23 78 17

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

# вывод результатов работы программы

print(f'''

Матричный шифр:

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

Введите матрицу в строку через пробел: 3 10 20 20 19 17 23 78 17

Матричный шифр:

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

дёьисжнбнжбеьнцмёаэгщсъттлюцгнхосцгфжгн

Расшифрованный текст:

время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

щвичнкфящёжщрээншусзуйдътюцбёэъяшщктыёжщжтийдмпжбзярюмсигфохжртичаужбвфэовкквкыкзчвзяжтиьудюяшгмювжаепыэсофрдюейхьёзпфрдчвзйсоюбёъштифчыхвлицхилщндкыбокттжжгнщъябллфпыдчикъллвусфвпъбёъётстрэуижлиатйчлихчрозюпдмзмлёъзпжюцбёввижгнйдмфшжтлйбншедщжгнншусзуйдътюцбёдёьбллющрюяшядмдбмйихюяшбээзмлфрдяпйшцпнфшшвкающмдуйюьсщигфъарыгхзчэдкхтщтяшцтиоььфчлчнкфящзфэхосжгнёжщюжръчасожцповусштдбнивщусзйдзшцпвфшьйотрцачиаъшцъйёгфрдэъпчияъжпшеьёттскиенаеэёгсюъвуефъацжщофабюязърцпяшлнкчтшюистщоъшхгчкгёядьчтйёьчеажщъёяъьявшзмлфрдяпйэдкхтщщплгйяоььъжхлжщцэоеяшъбьоиыфэцъаптъхйжгнзыгвядгобомиььхеёжкрмзпчичнкыфмшэомюцмээусзлеэзнкфмедщкфрдимежгнщцгмфвтэяхосймфыгщмнцьвпёдцщэобзчожтвнокдызпмвщвгщзыюжрътюцбёпчръщръмяюбёмллаьпяыогнобрчгюяшажиьгдмжзнкёчичыдгъяьиммсшмюцфммрэнохюёжюпплкяыщбяьъиохжиъожфцшщъщьачижччачижгншвквхшёсошязцезрэлцъсшдыоемъсимзецарцётьнцфжуэфгъмммвщебжюммвфудпулкктиэхосйбмтэячтфтщогземюцлиьвъжщцкбвжяьбнфжччхлтмъбёючжъяётгьеобэобеежржгнщпгхвдкнёыъчдбнмюцъммэуилэчфпыщчишюззмлбасзнкэучцпозбнжяымещкщтжвпуипоэяйщерджбйозюьщштщрэжюусзчйдчшрозжншрэнъенэъягнх

Расшифрованный текст:

вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходящийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпубликаций.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.номожноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднукартину.текстнатысячусимволовэтосколькопримернослов.статистикапоказывает,чтотысячавключаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблятьпредлогами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеизменновозрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиилибез.учетпробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименностолькоразмыразделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,таккакэтопустоеместо.однаконекоторыефирмыибирживидятсправедливымставитьстоимостьзатысячусимволовспробелами,считаяпоследниеважнымэлементомкачественноговосприятия.согласитесь,читатьслитныйтекстбезединогопропуска,никтонебудет.нобольшинствунужнаценазатысячузнаковбезпробелов.

## 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 = str(input('Введите 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

# вывод результатов работы программы

print(f'''

Шифр Плейфера:

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

Введите ключ: ключик

Шифр Плейфера:

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

зцднэйрурскююдaкпуюкаэунмбоукгкпгааёсичы

Расшифрованный текст:

время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

аруртюндстодпaкмдояогготдтаркпдрилaртаптодупкйтёбкнуылчмудчормрупрщдёдббёурпбщтаaшчмбиычвоупкгиораворадюмувуунщдюкёддбмюёещлюкбиbйавпкяхлщкмкфтзвфктнньфрожбклвфчмсиюаодафтщактувуаеафаьгбдуапкбаеерщлюкусжфбвёвъвдюпаякёутакмрпвмбдпкрафасикётёпёёулдвнунассикёдояогготдтаркпзцактёнуатгбёучтрпббёргбпaтакмкюдмгблючидктауъаёсукмоуилудчоумдояогготдтаркпзвупочпкылпртюндузпткпдрилтудочтщдаёрпаёёэгбдужрсиупоaшгкгаюючкгдугравгшупсьпaеашгщдюкеггущдпчратсаеунзгбккикмaоилёужрудпчюмкпорпуувжбaсбхувбисадёюмрутпврдёиивтседткигосaтдувикёетакмкюдмгбтчздпквёруупафюккгтуарпчраунюмндннрарёовтубаттилаюпрютгёудстафмггbудббёутулттюйbупшгдщдоaпяоггчтрспабкдёииюквамрилскдурспабкраребккиюдбаупехднудчоодрсдтвуёуёетуткюкеггущдтчздпкрадтнуёутупкылпсвёёaовмвбкbгйткпгбргпатаёbжтсптуовйурдтёсичоикодпaрспабкэёёавёиклкунючгьрмруодккёаaрпротупднгуупсиафемёауёакпупсbащктзaкдлпзюддмaстровгабиюдaётубгдщяпупдтптпaёвоaшгкстчздпкратрспвакбтджрутикодьспткбемкдгбзёaёьюдннуупёигкгурднуёуасартртюaсчaсичосакбтчудпяжрсидщдояпюкумяёудчопдвнаекмсапрспроочвёрумкиоуёавтедусикёпапкяхкмтуеруъзёвфнувёдояоггрнёеафгввнрспабкрасичы

Расшифрованный текст:

вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходящийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпубликаций.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.номожноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднукартину.текстнатысячусимволовэтосколькопримернослов.статистикапоказывает,чтотысячавключаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблятьпредлогами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеизменновозрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиилибез.учетпробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименностолькоразмыразделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,таккакэтопустоеместо.однаконекоторыефирмыибирживидятсправедливымставитьстоимостьзатысячусимволовспробелами,считаяпоследниеважнымэлементомкачественноговосприятия.согласитесь,читатьслитныйтекстбезединогопропуска,никтонебудет.нобольшинствунужнаценазатысячузнаковбезпробелов.

# D: ШИФРЫ ПЕРЕСТАНОВКИ

## 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 = str(input('Введите 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\_indx])

        for j in range(row):

            dec\_cipher[j][curr\_idx] = msg\_lst[msg\_indx]

            msg\_indx += 1

        k\_indx += 1

    null\_count = msg.count('\_')

    if null\_count > 0:

        return msg[: -null\_count]

    msg = ''.join(sum(dec\_cipher, []))

    return msg.replace('\_', '')

# вывод результатов работы программы

print(f'''

Шифр вертикальной перестановки:

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

Введите ключ: ключ

Шифр вертикальной перестановки:

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

вяпиовжчвтрзрвтыдеечмтлииетоа\_епиылнулкк

Расшифрованный текст:

время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

врртаяилчотчаьттомндядачооннлгнляоифанпииккееквбехтацонидоокоинтаяилеевилвоивюокичкаяилтоомовсикквзтсвчвяпдтдтоейиынтиуряроисадиамчоивматлсснморачоарйтнинсаыиоалзупеуиабттинслесоиноомзялсомсноктпеачнбпкэуеткаеофырисвиситозссосбмттодвыетатнвриклтзиьтттеорспкеекоивжеаянвпетоисьтчмокданлкезпаохщлреввтеиааинлхоцнукйвокроаодирбебонзлкнжбечтчмокнасьадлачданкстчмоолпесттсааапоякассяеививдвчтоезпетегзомрмсиидлавзоитлееваекпйсдеопячтссбмитчрлвчеъеамотисилмолрыдеовдптсмсарлкиеятктсмооккрииждпевттосаяилпеисаснаммочвооиясаептснебдгокттбтнлнуннтчабрлчпеанссоттсомнйстиьодйкотририаиилбшнмохлцчаттдытеуихаичдогвчмозонссормооооьндлиуттенссоэккиноктиоытчыаютбоьяислрелнкплотлплмтзигчяеандилтоеовзнзтткреояьтиоттчрлиектбвлвомсрраививвнткзалсабыоавчиьбззиюзакпосчннтемивталмвсмьыуввоапипеенлнксноптчгиьчтийсзнпузиндчбштуцзсзозбв\_тмтиыуввэотоеикптлпоияткаветмзхдеьириыбаттмсебелвлёзвыопаотонегкыувводнпзтииканруттыуввсьррлчатпзеттчлееттслесснеичзслобьдапюиуитрниисопкчвоиноствиткелсртиьяпеибчеооеитекпенодтмоесьаремвонррттчтоыаклттаотетдооырбиярдыаьиттчморлзчялижэемеегсятоссталыкеиопаноутоьснааыукеоок

Расшифрованный текст:

вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходящийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпубликаций.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.номожноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднукартину.текстнатысячусимволовэтосколькопримернослов.статистикапоказывает,чтотысячавключаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблятьпредлогами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеизменновозрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиилибез.учетпробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименностолькоразмыразделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,таккакэтопустоеместо.однаконекоторыефирмыибирживидятсправедливымставитьстоимостьзатысячусимволовспробелами,считаяпоследниеважнымэлементомкачественноговосприятия.согласитесь,читатьслитныйтекстбезединогопропуска,никтонебудет.нобольшинствунужнаценазатысячузнаковбезпробелов.

## 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'''

Шифр вертикальной перестановки:

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

Шифр вертикальной перестановки:

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

векаовртетмячзплткпривиынелждутчивыелои

Расшифрованный текст:

время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

чалекэтанвоькийотттперкдсимыетосрсстзатятчоучпнсаимвоомлттьиовтнолимваряаопгазионтщиамхвиалиьлвинойнитедплряндекоартотичлдхоектдиерейтчадянкобыквецбаоевтльишбтаихокоомнинтыелкхнпублстиефоркацмдожнзагцоедибезонвеуехглоилвитовтриочктоёбчыккнчхноодоминабзнпоалвакьзоелатючаивоыксдянаучуосктьиммоадиевнндирлоованоидитловспоровтчлькякинстатоиучтсчтпиктуекрикссмаеритмнвопсонловэлотоатыосксядвестосскатислпозяыввсяраечтетьзпаддесвткялтниюччаетливетотсебыясоютьптзйвамиирдеелриуегчиснидлытломгаичзмлиизокупотптрчнозеблптнеиокоизасменнлотлявмоизириерчечидсатввнаоссслиамволоватодизптниысяприрчаеиспрнотстбчеялквккатоопосмчийидтеаитяртелььтниайтостебнонъемеалистоитлблеиздевтчокексучвтсапуевреитмлтичивераипроетобямпремсоосистралнмлвсотовлоьвоваимксмвооербаотдзнмыранызчносделтпеметтааскстотчккчкиотдкнатаьаакпрзэкточоипкуосибнелютобнелыятззиоствымрьекзатыссожтяочтурыиесавфивиитьирдсятмотмспраимввыибедливачеажнтсолтвеныносвосгмопрчовэлбеиеометланятсопаослемкдпмизниептогоныйтприропутсяеткиаезятсчпксксьттбезопзтендчгитатиньиласслииевбеназозктпробаеолнлеотвбуьдтысетшяччуитнзснкатчвунукож кнонацб

Расшифрованный текст:

вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходящийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпубликаций.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.номожноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднукартину.текстнатысячусимволовэтосколькопримернослов.статистикапоказывает,чтотысячавключаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблятьпредлогами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеизменновозрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиилибез.учетпробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименностолькоразмыразделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,таккакэтопустоеместо.однаконекоторыефирмыибирживидятсправедливымставитьстоимостьзатысячусимволовспробелами,считаяпоследниеважнымэлементомкачественноговосприятия.согласитесь,читатьслитныйтекстбезединогопропуска,никтонебудет.нобольшинствунужнаценазатысячузнаковбезпробелов.

# E: ШИФРЫ ГАММИРОВАНИЯ



## Одноразовый блокнот К.Шеннона

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

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

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

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

**Код программы:**

import random

from base import alphabet, input\_for\_cipher\_short, input\_for\_cipher\_long, 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,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

[12, 47, 30, 29, 1, 28, 12, 13, 17, 28, 14, 8, 8, 13, 23, 25, 7, 14, 25, 7, 49, 56, 27, 11, 13, 16, 10, 24, 9, 14, 3, 26, 22, 6, 9, 6, 7, 0, 6, 5, 7, 31, 7, 12, 12, 11, 1, 15, 32, 25, 24, 4, 31, 17, 7, 9, 2, 8, 23, 9, 17, 17, 48, 4, 3, 19, 1, 10, 1, 4, 27, 9, 25, 23, 15, 17, 29, 12, 13, 3, 47, 21, 22, 12, 19, 19, 58, 10, 8, 12, 11, 14, 31, 31, 31, 9, 6, 18, 1, 49, 3, 10, 27, 3, 19, 21, 26, 16, 13, 2, 5, 21, 15, 26, 5, 5, 5, 32, 29, 28, 9, 14, 25, 3, 22, 15, 6, 14, 34, 14, 7, 15, 18, 0, 29, 1, 27, 14, 12, 46, 16, 27, 2, 4, 27, 16, 16, 10, 12, 0, 21, 1, 28, 24, 17, 1, 8, 27, 19, 13, 11, 16, 20, 9, 1, 34, 23, 19, 25, 7, 20, 11, 1, 14, 23, 9, 10, 21, 4, 10, 29, 6, 3, 6, 6, 9, 23, 0, 4, 0, 14, 14, 9, 12, 15, 1, 20, 14, 27, 5, 20, 0, 13, 13, 25, 27, 1, 7, 7, 2, 15, 6, 7, 27, 10, 23, 2, 9, 31, 21, 22, 25, 22, 23, 36, 23, 31, 10, 18, 8, 19, 28, 21, 8, 4, 23, 1, 22, 29, 31, 28, 27, 31, 22, 21, 3, 23, 30, 7, 11, 0, 18, 31, 20, 4, 20, 25, 27, 19, 8, 45, 26, 9, 14, 7, 16, 20, 15, 23, 26, 11, 16, 29, 23, 7, 24, 8, 4, 4, 9, 9, 14, 22, 19, 15, 23, 3, 2, 15, 9, 19, 1, 9, 11, 9, 17, 14, 27, 12, 25, 7, 21, 2, 8, 31, 13, 23, 44, 13, 5, 9, 29, 23, 20, 11, 25, 2, 5, 2, 4, 26, 23, 20, 27, 9, 27, 30, 24, 14, 7, 17, 23, 4, 0, 13, 17, 33, 22, 10, 20, 9, 13, 24, 16, 27, 1, 3, 11, 21, 9, 17, 20, 17, 20, 19, 26, 21, 24, 1, 17, 0, 9, 22, 9, 19, 4, 44, 22, 25, 28, 26, 28, 5, 29, 12, 3, 12, 8, 22, 18, 5, 7, 14, 8, 27, 16, 12, 15, 29, 30, 6, 20, 17, 29, 15, 14, 18, 24, 7, 22, 28, 38, 15, 28, 6, 16, 4, 6, 21, 23, 9, 22, 9, 1, 18, 26, 52, 22, 19, 21, 25, 37, 15, 28, 31, 8, 30, 36, 19, 17, 14, 12, 26, 5, 4, 30, 6, 0, 5, 25, 0, 14, 25, 27, 1, 36, 14, 15, 6, 11, 31, 13, 17, 1, 31, 27, 28, 7, 12, 22, 8, 28, 17, 22, 16, 25, 25, 28, 21, 29, 13, 0, 20, 17, 13, 23, 23, 37, 15, 8, 58, 4, 8, 9, 27, 7, 5, 19, 21, 13, 21, 25, 24, 19, 4, 2, 26, 25, 12, 11, 24, 26, 8, 19, 11, 11, 31, 35, 1, 9, 26, 22, 2, 25, 26, 60, 24, 2, 49, 15, 22, 27, 14, 3, 29, 12, 7, 13, 7, 31, 2, 12, 22, 13, 14, 17, 5, 6, 31, 21, 21, 14, 14, 25, 4, 7, 30, 25, 27, 10, 8, 16, 7, 14, 6, 29, 29, 17, 2, 31, 15, 12, 14, 19, 17, 31, 21, 25, 16, 10, 26, 19, 9, 31, 22, 15, 32, 18, 1, 27, 3, 0, 17, 5, 47, 10, 11, 16, 4, 29, 4, 4, 17, 21, 10, 14, 10, 10, 13, 50, 15, 15, 2, 19, 15, 10, 18, 7, 29, 48, 13, 15, 13, 42, 7, 1, 13, 7, 0, 27, 4, 0, 16, 13, 17, 4, 34, 1, 12, 7, 6, 8, 19, 13, 10, 9, 22, 25, 10, 23, 12, 23, 30, 29, 40, 10, 16, 2, 16, 2, 7, 13, 31, 27, 15, 16, 11, 0, 19, 4, 10, 31, 26, 16, 24, 56, 2, 18, 8, 0, 30, 24, 5, 11, 0, 6, 31, 19, 10, 31, 8, 23, 5, 24, 26, 6, 30, 11, 13, 25, 16, 17, 13, 6, 30, 21, 15, 7, 1, 9, 2, 15, 12, 14, 0, 10, 6, 31, 5, 29, 26, 5, 30, 12, 28, 4, 18, 8, 1, 13, 24, 24, 7, 8, 21, 27, 25, 17, 18, 2, 27, 16, 3, 4, 12, 22, 45, 15, 0, 10, 14, 29, 34, 31, 12, 22, 31, 21, 26, 2, 16, 16, 11, 15, 0, 6, 23, 26, 25, 2, 2, 28, 16, 3, 24, 2, 25, 17, 26, 10, 26, 16, 21, 25, 27, 4, 18, 21, 23, 6, 0, 18, 30, 6, 24, 17, 1, 16, 25, 22, 20, 15, 30, 37, 29, 4, 21, 54, 6, 18, 3, 8, 6, 12, 23, 27, 15, 27, 4, 51, 29, 30, 7, 14, 31, 5, 0, 18, 18, 29, 31, 5, 27, 19, 21, 30, 4, 4, 21, 31, 31, 46, 3, 16, 18, 4, 12, 25, 5, 22, 28, 7, 11, 9, 29, 23, 29, 22, 11, 12, 10, 7, 24, 29, 35, 7, 13, 10, 12, 12, 17, 17, 1, 14, 17, 22, 19, 21, 20, 16, 0, 20, 17, 51, 13, 30, 10, 24, 2, 25, 20, 21, 22, 17, 13, 23, 13, 25, 29, 41, 25, 20, 26, 23, 25, 12, 20, 1, 26, 34, 11, 16, 20, 29, 20, 17, 44, 5, 21, 13, 27, 7, 17, 50, 21, 0, 13, 31, 53, 31, 27, 21, 19, 26, 24, 17, 41, 5, 30, 21, 8, 14, 16, 45, 3, 22, 37, 17, 21, 20, 25, 12, 45, 26, 14, 14, 1, 50, 18, 21, 22, 5, 18, 24, 17, 24, 13, 28, 21, 10, 16, 6, 52, 1, 30, 35, 17, 23, 2, 25, 1, 20, 19, 9, 20, 7, 18, 26, 3, 9, 11, 31, 11, 22, 26, 7, 11, 15, 16, 31, 20, 8, 11, 20, 11, 5, 0, 27, 24, 1, 9, 30, 22, 23, 14, 7, 41, 28, 31, 16, 30, 21, 10, 28, 19, 10, 14, 25, 10, 2, 19, 21, 27, 0, 11, 26, 26, 27, 30, 12, 21, 4, 10, 0, 10, 25, 13, 11, 20, 13, 31, 19, 27, 15, 29, 22, 8, 21, 31, 21, 11, 9, 0, 15, 10, 7, 4, 17, 14, 16, 24, 9, 22, 3, 55, 17, 3, 6, 46, 23, 21, 14, 18, 18, 1, 4, 18, 25, 25, 0, 16, 7, 8, 19, 51, 25, 30]

Ключ:

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

Расшифрованный текст:

вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходящийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпубликаций.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.номожноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднукартину.текстнатысячусимволовэтосколькопримернослов.статистикапоказывает,чтотысячавключаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблятьпредлогами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеизменновозрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиилибез.учетпробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименностолькоразмыразделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,таккакэтопустоеместо.однаконекоторыефирмыибирживидятсправедливымставитьстоимостьзатысячусимволовспробелами,считаяпоследниеважнымэлементомкачественноговосприятия.согласитесь,читатьслитныйтекстбезединогопропуска,никтонебудет.нобольшинствунужнаценазатысячузнаковбезпробелов.

## 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, sbox):

        self.\_key = None

        self.\_subkeys = None

        self.key = key

        self.sbox = sbox

    @staticmethod

    def \_bit\_length(value):

        return len(bin(value)[2:])

    @property

    def key(self):

        return self.\_key

    @key.setter

    def key(self, key):

        self.\_key = key

        self.\_subkeys = [(key >> (32 \* i)) & 0xFFFFFFFF for i in range(8)]

    def \_f(self, part, key):

        temp = part ^ key

        output = 0

        for i in range(8):

            output |= ((self.sbox[i][(temp >> (4 \* i)) & 0b1111]) << (4 \* i))

        return ((output >> 11) | (output << (32 - 11))) & 0xFFFFFFFF

    def \_decrypt\_round(self, left\_part, right\_part, round\_key):

        return left\_part, right\_part ^ self.\_f(left\_part, round\_key)

    def encrypt(self, plain\_msg):

        def \_encrypt\_round(left\_part, right\_part, round\_key):

            return right\_part, left\_part ^ self.\_f(right\_part, round\_key)

        left\_part = plain\_msg >> 32

        right\_part = plain\_msg & 0xFFFFFFFF

        for i in range(24):

            left\_part, right\_part = \_encrypt\_round(left\_part, right\_part, 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(l) for l in itertools.repeat(list(range(16)), 8)]

sbox = (

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

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

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

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

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

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

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

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

    )

key = 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:

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

{encode\_text\_short}

Расшифрованный текст:

{output\_from\_decrypted(decode\_text\_short)}

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

{encode\_text\_long}

Расшифрованный текст:

{output\_from\_decrypted(decode\_text\_long)}

''')

**Тестирование:**

/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab05\_14\_gost89.py

Гост 28147-89:

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

56754026145183696286056690583114096463305996216823972540084957071450361516686534633542086281498134144445869433834001779406256238181699835567875396711529074587560012517759518637140963090682

Расшифрованный текст:

время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

2660504179384763560870172300148736432061604479961145151290917749597470334059438903438971428697044508153936836050236848738880049098296804522950287185469780216499411778268801517322745336438645323207966302724507468373474684283063682350895571195422576014055891201234951917117574410777206443142305502703477508839984518986714583143400164257550139955951705291905260249349753577193635641327610319052903559952970011424379847993621475968548034595715017650775261104984599500375814441113013017875969917042030848511398372774550996207392191339181039778867511513245777658070846405184637307157764373019723472702707385419563060306821034731763551154503824714264812313316130706264074393244483028493575194280183695841632846694677441664704015987166764373865851781422124591521492824680316201781243695172538536956831291069312772130538659077026628200178989017827102455811284644197446147728227370823795695828448846841904902145096572681358073870106763470339835923246436026070889038200287505359250682058240850262033170450161709526251248889202783539456604116848059039836482794508570635524117505841700390724701975258248279406107572222767582208189240451665976152382603851822752929335645908960132784104671236312325136002226758406236192110306294400818405951215215170099869890409677082940113502073956343581542790367957821887506781091409598967321936432916690830836157099844286928789002776683458441918340765274704251161726938975327301299827924533461687715964034781601529646429568161764717204759432825169398902206756041104866390413896194971752446779789219834868520961790344530438359259087681089419095818947367924242461052662246177551436759868060964413164161379298518749743822796623991927620430286913534792524472501385616062283433788266136034182441382596852828678815464248898242655956549987693016474857317967045765805402303929238917825768453421398261020477598060009464910092710806727389232375505337101980972052186720533409389553482212142338388825165064316873200362417005979383125608949135305078202880494402590429918423534867498481325002725186208672068804910219444128725412313172857315960092225568479643477408050023889229147527678395405300014514788559927920823504740092277359994838446144638292221204810273531732178907255974872373943777174489278720360030245513206396286680923132665591932957479381688881713391447550197512421414080155618078837436716254239495738949352604395115116413190410600237037255778849942318131400487867993716612245815508666788386611535983488700840787806608061738649595754409762598103992609851236390147034058997120081926015140106802936589820873919900023139596870752250532835059504769730299189404268942453718027773910730364364478309783557301001910299876928643382267779773297344160509407488050435104053968601803987961053508330097865866726566439196742557135164782260908456730128332911775983685320723472682806496460888618031721287732914731095065151066593148360400366560924904292376661781984325986153706049185968639527031555339328097086138493583066405661508631851879253262157549893336342085850379421169485707846412390764082738059775423863152365222207358370420848642329032896143477728147462873088301241858500518352670460459172294780626495853313915274281775969597939390408588921013191052753639133045535705372177150216764101504465020904745448395363206321778018466643874865465360874268566642305436776611825466962196427195722777111044656000900826008555466570413212540474867009345483678501394846081885654740922361570249280181151517115435307624861660026083828186182751031813699667814905465529350145221133432739448140980614569357618476483550862855556317816602783334176925406116449091395553726830899141742583735706859991379762269375783746468934694732394565580644564403759572656563966408160663598247766009685660723020552419971761400822686542373391732314068592807190033601276697644910836577636698226330648236553701481899448499076055004435636081490697301414470015727863530449650719384883208173939823273509148707571491685195572982448679697231841618128665142056093605453472388015139879689507614636267932199613494818601175694495106823965735679852248453340323322884737319847422163663683329727707507865218141639579913178721468797011563675058848404432420337436492844035343087300285655784488990284687493681659591330992349220652565296410779661349740175153061305288645417558786186533328282877556855948224417771287356978869466410912431116815452381165518582443064263928368461040647310768425435788168185304791521567591202904460425598689779839465289865075045548803625879099049880457756634096216287517583379850827114094281279725186959719047716798991463990201696001595394956661030446430788409672457741058597704420444123772447654933964197877421550863353732634744480181841548898737461517584844294120903204988150464053418061288143949058682199344504008861383378392402642556140683240706965932049456398214008601555594844595489325508401028823872678584029686001711761573768314347772017072852998336991250377282698317207452140810115763062272712262299394159864510307804974680262082930993792882105434472845097445767818724620636250333319570116562701500864340123631182252024978565591020490325434070533337461974698559903223030221449025366488862781604023577829136213595700371095616666191295280158427548094843204106432381377975693276853993148974688913227746868482199358865760126313652003609854882178324694

Расшифрованный текст:

вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходящийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпубликаций.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.номожноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднукартину.текстнатысячусимволовэтосколькопримернослов.статистикапоказывает,чтотысячавключаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблятьпредлогами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеизменновозрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиилибез.учетпробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименностолькоразмыразделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,таккакэтопустоеместо.однаконекоторыефирмыибирживидятсправедливымставитьстоимостьзатысячусимволовспробелами,считаяпоследниеважнымэлементомкачественноговосприятия.согласитесь,читатьслитныйтекстбезединогопропуска,никтонебудет.нобольшинствунужнаценазатысячузнаковбезпробелов.

# F: ПОТОЧНЫЕ ШИФРЫ

## А5 /1

А5 — 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

# 0101001000011010110001110001100100101001000000110111111010110111

key = str(user\_input\_key())

set\_key(key)

print(f'''

A5/1:

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

Введите 64-bit ключ: 0101001000011010110001110001100100101001000000110111111010110111

A5/1:

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

110110111010111010100001110101110010000011100101011111001001110110010010101100000101101110000111110111101000110110110011011100000100010011000101110011111100110001110110110111110101101011100011111101001110011000001001101010010111101100100011001101010010111010010100100010001000010101100101001011111011001010111110001000101001010011111101100001100000010100001001101001100010000101010111100110000100110110111000011011100100001000010001000011010111011101100111101011111000

Расшифрованный текст:

время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

110110111010111011011111110100000101000011100110011111000110110110011101101100000110101111110000110110010111110110110010011101111110010011001110110010000110110000001000110110000110101011100110111101000000011000001001101011100111101101011010001100101111111010011000100011110110010100010001001011110101001010110110001001011000010010000001100000011100010100000010101001101011000100101011100111110001110110110111011010010011001001100011000010101011011100010100101011111100100110111000000101001101100011111011011110011101111110101101001011101000101010111010110100011100110000110111011001110010010001111101010000101101101000111100011011100001111110100111000101101000101001110010101100011011111110001111001100010110001011000000101101100010011100100001110010010010111011101010100011001110110111000000010100111001110101010011110000101100000100111101100000100010001111111111110011010000110000000001101110000000010110111100001011011111000111001001001001011000111011000100000000101011100110101010111111110111100101100111010011000010010111100000110101010010010110110000101010100100010101110100010010000000001010011010110010110101110101111011010101101000001101111100011000101100110001110110111100000010000000011001101000111100110010111110111101101111100011000010011100000011010111011011100010101100110100100101111001110000111010111010110001001011011111101011100011110000101000001010000011000100010101001100100000100101111000111100000000010100000001011001110111000111011010101001100110011100111011100101000101001010011011001011000110001101110001110110110100000010101000111110100100110011001100110011011010000101011011010111100100100000110000100111011001001110000100010011011110000110010010000000000101111010101110001000110101111001011100110001101011000111100011000000010001111010110001100111001111100000010011101001000101011001100101000010010010111010010101011000100111011000100010111110111111110110000010100100000111000001010000110001011011010010001110011111111101011000011000111110000011100101011101101011111110100111011010001001111110000001010101011100000011011111010110000110111101100110100000100110000011111001010110101111000001110111011000000100111101001010000101100101011001010001100111111010011110101011001110111101110101110010101100011100000011010011011100110101000110111000101000100001001001100010000011100111111011011001011010100001110011011001101001000000111100000100000001001010110011001001111110001111001101101010000100001001010010001110100010101001011001010111110001011011100100010110011100100011101110101101010011001101110111010000111100000011010100110100101101110010101101000000000101101000100110010111011101010100101110100101010110011101011011000001101001101101010111010110011111111000001001101011111000101101111010010111010011111111001111010111100111101001110101010000000011011100111000110010010100001111110010011011100001100011100010000000100000000110001011010111110110000000001010000001010111000001010100111001110010100010011011000011111000000011110110100000101001101100111001000010110110111001101111011000011100110011111100010101100010101101011000110110001010110110101100111111010100000101111011101111101001101111011011000100110100110011001001010011101000110101010101110100101100010011111011010101001111010100111100010001001110110010100011011001011000110111100111100010110010010001000110100101111110100111001001110001100011000101011111010111101101001101001011011100001111110111111011001101000011001001001100101010011101110010011111101001000100110110011110010100101010000000101101011010111001110110110100001000000100000011010111010001111111110100101110101111011110000111000000100011111100010000000110100000101010001101000011110010001000111100011000111001000111110010010000101001111011100011111000010100010111111010111011000101011001011111010110101011011001110000001111010000101011110100011001010011100000010011011110110111100011101001010111011110110000111010101010110001111000000010100100110010101110001101101110000000000101001111000010010111001010101111111110100011101000011001101110101011101110001110101011101010110110111110001010101100110001101011101111101011011011011011010011100000001111111101000101010010110001011111000010110111000010011001010010110100111011000010010010010011110110010110100100001001111100101001101111010011011101010101100000100001011100001011001000000001000110101110000011100011011011000000110001100011001110010001000000100011111001100011000011011010011110000000001001001011100000101000110110000010001100011001000010011000011100101100011010110111010001110011011000110011100001001001011010110110101011001110011111000101100011000110000101101001011100111100010001101001110011010110111100110001000111110011001000101001100010101101011111100010011001000100001110000111111010011011111001001001000011110001000011000100100110000001001111001110111001110011011001010100000010101001110000110010111000001001110111000001100100100000001010011111011100101000110000110000100011001101010000111111000111101000100101110110100000001000011110111110011000010110000100100110011100001001100101001110000111111101011000001110100000100101101101010001000111111001100010011011100110101011110111010000111110111110111010101100000110001110000001110001001001101001011111011100111010010100101110101111110001101111110110110111011111000000100101011010111110111010000110101010100001100101111011001000000111100110000011111101101111100111000111000001100110110010010101110000000010000111001100011011011000110101101011011100000001110000000110111010110111000101111001010000101010010010000000111001001101010101000101011001000010110011111010001110100110011010110111111000110000010010100110111110100010000010100111010110011111000001111110001011110010100101011011011100111100000010000101100101000100010001011011010111010110110111010010110101100001100111101011010000101000000000100101100111100101101011100001011111010011011000000101001111110111001000100000001111000001011110100111011101000101101001101001110101010011110111100000100001100100111111101101010110010101001110001110111000010011011000110011000011101000001000110111110010010101001000111111111011001011110110110000011011100111011101001111000110110111011100101011010101100110011010000010010111001010101101110101011101101101111001001011110100010101011110011001010010010111000111010110000011111111111001111100100000100001111101000110000010111000111001001010010010010101111101110111011001011010101101100101100001001011111101000010001111100000110110110000010011111100001010001010000101010100011110111000001110000100101101010111100010111000010010010000011000001111011111100011111110011011101010011111111010101101110101100110011100110100010011110111100010010000010010011110000101011100101011011100111000000110010010100011000010101101110011101011111010000000010010001111011101001000110110111001111001000011011101001001100110100000111010011010110000000111011111100011011010010010011000010000110101001110000101110100011011000000100110001000111011001101100010100111111100000000010110111011010011101110001010100010000111100011010100000100101110110001000110010101111100101101000011010010011000111011000010001100001001101100000100000110101001100000110010010011111011011100111011001001000111111011001011101101011100100100111110010001001101110110100100011110000100011101100111000000011011000011110001110011010000001011010100000110011110100101110100001110111111101000101010110011101111101100111000110001001101001001000000000001110010011111010110000011010010010100110011111000000101010001101100111101100010111010111011000110001001011000110011110100010011010001101101011001000101111101000101001010100011011111011011000011010010000011111110011000100100101000000100101000011100111100001110110011111001000111100110000001010001100001011011100110011101100110100000000100101001111011101111110000100100011000011011110001000011110110100010100101101010000101001001110001011001111000101001100010000010100001011000101100101100101011000001110100001101001100011011110100000000111011100001010001101010110101001111100000101110010011110110000100010011110001100010010100111100101100001011011000011001010011110110100111000000111100100111001001010100000111101010010100111010101001100111111100001000110101101100000110100010100110011011111011010101001110101011000001000011111000001001110000101010010000000111000101101110011110100001001001111011111010101110001001010111111001110001100110010110100111000001110010000111011001100101111001000000111011110011000011001010101000000010110010100001011010101111010101010001111111011101101001000010011110111100011100011100111010100110101110011101101101111111001110011010100001010101110100011110111011100000110011010110000001110100011001100110111010100010101011001010110101110000000010001001110010001111010001111000010000100010000111100101011010001100110010001010010000010101001011101111111111011011110110001101010010011110101010101001011100011011110100110001110101100000110111000000100111101011111101001000011000110011111000001010110110011001010010001010101010111011110110100101000010111010101110101101001000001000000110101100000101101100011000000001001011101101001001111111001011011001100010011101011010000001111000101100000101010000111100100111101101001000000000111000001100001101001000011011111010101100111110111010111111111011001111100010111010000010111111110000011000001001111110101000001100110111101011101101010011000111100111111011100001011000011001001100000000001111011111001010010101011001110010010000110011001011111001110011011111000000011000001000001001000010001111101011100011100100110101101110001000100101100100110000000001000111111100001110011001111111111010010110110010100000010101000100000101010001000010011100100110110111000010011010110110110100001000011110111000110001000011101000101100000000011001001111111101001111100100001110001010000010110101111010100010000111111010011010101110100001001010100110010100000001100011100001110011001101011010000000010100010100010000101111011011101101000110100101110010010110111011010010001110110001111100100010011111110000000110000011000010111001000100111001000100000011011101011110101101100110100110111011110000111110010001100000100001010101011100111001111010111110100101101000110111011111111001000110001000100111001100111010111100011001101001101101101101101001001101001111011111100000101001001110000100110101110011010001111110110000011110101010101000011101011111100110000000001010101011010110011100100111001001101100010101110101110100000010010001100100111000111011100001111000100111111001110000001000010000111001110001010001010000111001110110111000000100110101100111001110011110101011101001101101000001010101000011100010010010100000011011011111010101101100101100001010011110000110011000111111010000100100111000111101100110101010010100000111000010100101001110101010000011011100110000010110100010011000101000110001111110100101111010000100100010001110010110101000111011111111000011011100100111111110101010111100000010110000111011100001010011010101000000010011001101101101010001101100010111010001111001000000100010000110101110111111100100100011010001100111110110101110011000001110110111011110100010110110011111000110111100100111011011100000001000011000010000011001001111010001111001001011011111111001011010010101111100010000101010100010000001010000101101110111111111101101110011000000000101110000011110011111111111110101110111010010001111000110010101010001101110001011011111001010011001101110011001111000110011001011001000100001011011011111100000000100011011110000110001001111111000001000100101110010100000000001010100000010111101100101111011010001010110011100100010010000011111100100010001111011001001000011110100001000001011001010111101100011011110010100011100100111110000110010001110110000001101101011111110101111101011111110011001011011101100111011100110111100001000001010101000110100000111101111001011000100011111001000111001000000110011101101010100111010100110001001101001010000010111111000111000010100010101010111010010110000010100110010011110111101010100011010010110010110110010100110010100000000010111001110101010111101100110101001001000010101100010010111010110110100110100000011010100000111100100000100001001011001101100011100010011011001011111011000000101111111111100010010110011001001111110101111010000110010010011000111010011001100111111110010101111110110000011000100001000100110100011110010101101010111000001001010001110000111001000011110111000111010110011110100001010001010101100010010001111011110010001011001001011001101100001100000010001110001011001101001100110000111101101000101001100010010111111100010010100000110110100111100011000111001011101010010101110001010110011111111101000110111000011101010010110001101110011110111000001001001001101010011111101001000111100010000010110111110111001000101000001100011110100101010100111001001000101111000100101100011001011101111001110100101010000001000000100010101101011011101001011000000111111010111000100101110001110110000101011001000010101100110000001101111001100111100000100110011111110011011100100011011110011011010110010011101010001101100111000101111010100000111001000100101101111000110001001101101000100110001000001101001110111010111011010111011101001001100110000110010011111101010101011011010

Расшифрованный текст:

вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходящийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпубликаций.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.номожноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднукартину.текстнатысячусимволовэтосколькопримернослов.статистикапоказывает,чтотысячавключаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблятьпредлогами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеизменновозрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиилибез.учетпробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименностолькоразмыразделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,таккакэтопустоеместо.однаконекоторыефирмыибирживидятсправедливымставитьстоимостьзатысячусимволовспробелами,считаяпоследниеважнымэлементомкачественноговосприятия.согласитесь,читатьслитныйтекстбезединогопропуска,никтонебудет.нобольшинствунужнаценазатысячузнаковбезпробелов.