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 4](#_Toc90823784)

[Постоянный модуль 5](#_Toc90823785)

[Блок А: ШИФРЫ ОДНОЗНАЧНОЙ ЗАМЕНЫ 6](#_Toc90823786)

[1. Шифр простой замены АТБАШ 6](#_Toc90823787)

[2. ШИФР ЦЕЗАРЯ 8](#_Toc90823788)

[3. Квадрат Полибия 11](#_Toc90823789)

[Блок В: ШИФРЫ МНОГОЗНАЧНОЙ ЗАМЕНЫ 14](#_Toc90823790)

[4. Шифр Тритемия 14](#_Toc90823791)

[5. Шифр Белазо 17](#_Toc90823792)

[Блок С: ШИФРЫ БЛОЧНОЙ ЗАМЕНЫ 20](#_Toc90823793)

[8. Матричный шифр 20](#_Toc90823794)

[9. Шифр Плейфера 24](#_Toc90823795)

[D: ШИФРЫ ПЕРЕСТАНОВКИ 29](#_Toc90823796)

[10. Шифр вертикальной перестановки 29](#_Toc90823797)

[11. Решетка Кардано 33](#_Toc90823798)

[E: ШИФРЫ ГАММИРОВАНИЯ 37](#_Toc90823799)

[13. Одноразовый блокнот К.Шеннона 37](#_Toc90823807)

[14. Гаммирование ГОСТ 28147-89 42](#_Toc90823808)

[F: ПОТОЧНЫЕ ШИФРЫ 47](#_Toc90823809)

[15. А5 /1 47](#_Toc90823810)

[16. А5 /2 55](#_Toc90823826)

[Блок G: КОМБИНАЦИОННЫЕ ШИФРЫ 63](#_Toc90823827)

[17. МАГМА 63](#_Toc90823828)

[БЛОК H: АСИММЕТРИЧНЫЕ ШИФРЫ 68](#_Toc90823829)

[21. RSA 68](#_Toc90823833)

[22. Elgamal 72](#_Toc90823834)

[Блок I: АЛГОРИТМЫ ЦИФРОВЫХ ПОДПИСЕЙ 86](#_Toc90823835)

[24. RSA 86](#_Toc90823837)

[25. El Gamal 89](#_Toc90823838)

[Блок J: СТАНДАРТЫ ЦИФРОВЫХ ПОДПИСЕЙ 92](#_Toc90823839)

[26. ГОСТ Р 34.10-94 92](#_Toc90823840)

[27. ГОСТ Р 34.10-2012 95](#_Toc90823841)

[Блок K: Обмен ключами 100](#_Toc90823842)

[28. ОБМЕН КЛЮЧАМИ ПО ДИФФИ-ХЕЛЛМАНУ 100](#_Toc90823843)

# 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

def atbash(input):

    return input.translate(str.maketrans(

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

print(f'''

ШИФР АТБАШ:

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

{atbash(input\_for\_cipher\_short())}

Расшифрованный текст:

{output\_from\_decrypted(atbash(atbash(input\_for\_cipher\_short())))}

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

{atbash(input\_for\_cipher\_long())}

Расшифрованный текст:

{output\_from\_decrypted(atbash(atbash(input\_for\_cipher\_long())))}

''')

**Тестирование:**

/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab01\_1\_atbash.py

ШИФР АТБАШ:

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

эоътачпмпоцуцэдцрмуцэдсъшылмзъурэъфямзф

Расшифрованный текст:

время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:

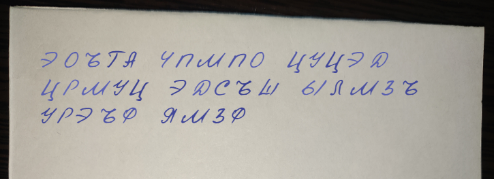
Зашифрованный текст:

эрмпоцтъонмямгцсямдназлнцтэрурэмзфвмрырнмямрзсртяуъсгфцхмъфнмчпмрпмцтяугсрпрыйрыаёцхыуафяомрзъфмрэяорээцсмъосъмцуцтяьячцсяйцуцыуасъюругжцйцскротяицрссдйплюуцфяицхмзфэмяфртмъфнмъоъыфрюдэяъмюруъъыэлйцуцмощйяючяиъэцрюдзсррыцспрычяьрурэрфмзфсртршсрцюъчсъьрмзфсямдназлнцтэрурэоъфртъсырэясрцнпругчрэямгрыцсцуцыэяфубзяцрыслфяомцслмзфмъфнмсямдназлнцтэрурэвмрнфругфрпоцтъосрнурэмзфнмямцнмцфяпрфячдэяъмчпмзмрмдназяэфубзяъмэнъюанмрпамгыънамцуцыэънмцнурэноъысъхэъуцзцсдмзфсрчпмънуцчурлпрмоъюуамгпоъыурьятцчпмнрбчятццыольцтцзянматцоъзцсярыцсцуцыэянцтэруячпммрфруцзънмэрнурэсъцчтъссрэрчоянмяъммзфэфрпцояхмъонфрхыъамъугсрнмцпоцсамрнзцмямгмдназцнпорюъуятццуцюъчмзфлзъмпорюъурэлэъуцзцэяъмрюеътмъфнмяпоцтъосрсянмрцуцыэънмцнцтэрурэцтъссрнмругфроячтдоячыъуаътнурэянэрюрысдтпорнмояснмэртмзфнзцмямгпорюъудчяфячзцфцсъубюамчпммяффяфвмрплнмрътънмрмзфрысяфрсъфрмродъкцотдцюцошцэцыамнпояэъыуцэдтнмяэцмгнмрцтрнмгчямдназлнцтэрурэнпорюъуятцчпмнзцмяапрнуъысцъэяшсдтвуътъсмртфязънмэъссрьрэрнпоцамцамзфнрьуянцмънгчпмзцмямгнуцмсдхмъфнмюъчъыцсрьрпорплнфячпмсцфмрсъюлыъммзфсрюругжцснмэлслшсяиъсячямдназлчсяфрэюъчпорюъурэмзф

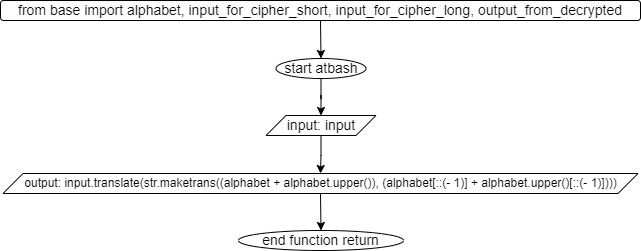
Расшифрованный текст:

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

**Карточка:**



**Блок-схема:**



## ШИФР ЦЕЗАРЯ

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

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

**Код программы:**

from base import alphabet, input\_for\_cipher\_short, input\_for\_cipher\_long, output\_from\_decrypted

import random

key = 5

def caesar\_encode(input, step):

    return input.translate(

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

def caesar\_decode(input, step):

    return input.translate(

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

print(f'''

ШИФР ЦЕЗАРЯ:

Ключ: {key}

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

{caesar\_encode(input\_for\_cipher\_short(), key)}

Расшифрованный текст:

{output\_from\_decrypted(caesar\_decode(caesar\_encode(

    input\_for\_cipher\_short(), key), key))}

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

{caesar\_encode(input\_for\_cipher\_long(), key)}

Расшифрованный текст:

{output\_from\_decrypted(caesar\_decode(caesar\_encode(

    input\_for\_cipher\_long(), key), key))}

''')

**Тестирование:**

/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab01\_2\_caesar.py

ШИФР ЦЕЗАРЯ:

Ключ: 5

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

жхйсдмфчфхнрнжанучрнжатйлишчьйружйпечьп

Расшифрованный текст:

время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:

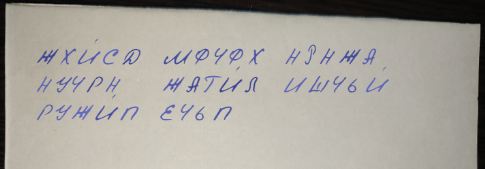
Зашифрованный текст:

жучфхнсйхцчечбнтечацдьшцнсжуружчьпвчуиуцчечуьтусерйтбпночйпцчмфчуфчнсербтуфуиъуидюноирдпехчуьйпчужехужжнтчйхтйчнрнсеземнтеънрнирдтйёурбэнънтщухсеынуттаъфшёрнпеыночьпжчепусчйпцчйхйипуёажейчёурййижшънрнчхкъеёмеыйжнуёаьтууинтфуимезуружупчьптусултунёймтйзучьптечацдьшцнсжуружхйпусйтиужетунцфурбмужечбуинтнрнижепргьенуитшпехчнтшчьпчйпцчтечацдьшцнсжуружвчуцпурбпуфхнсйхтуцружчьпцчечнцчнпефупемажейчмфчьчучацдьежпргьейчжцйёдцчуфдчбийцдчнрнижйцчнцружцхйитйожйрньнтачьптумфчйцрнмрушфучхйёрдчбфхйирузеснмфчцугмесннихшзнсньецчдснхйьнтеуинтнрнижецнсжуремфччупурньйцчжуцружтйнмсйттужумхецчейччьпжпуфнхеочйхцпуоийдчйрбтуцчнфхнтдчуцьнчечбчацдьнцфхуёйресннрнёймчьпшьйчфхуёйружшжйрньнжейчуёяйсчйпцчефхнсйхтутецчунрнижйцчнцнсжуружнсйттуцчурбпухемсахемийрдйсцружецжуёуитасфхуцчхетцчжусчьпцьнчечбфхуёйрамепемьнпнтйргёдчмфччеппепвчуфшцчуйсйцчучьпуитепутйпучухайщнхсанёнхлнжнидчцфхежйирнжасцчежнчбцчунсуцчбмечацдьшцнсжуружцфхуёйреснмфчцьнчедфуцрйитнйжелтасврйсйтчуспеьйцчжйттузужуцфхндчндчьпцузрецнчйцбмфчьнчечбцрнчтаочйпцчёймйинтузуфхуфшцпемфчтнпчутйёшийччьптуёурбэнтцчжштшлтеыйтемечацдьшмтепужёймфхуёйружчьп

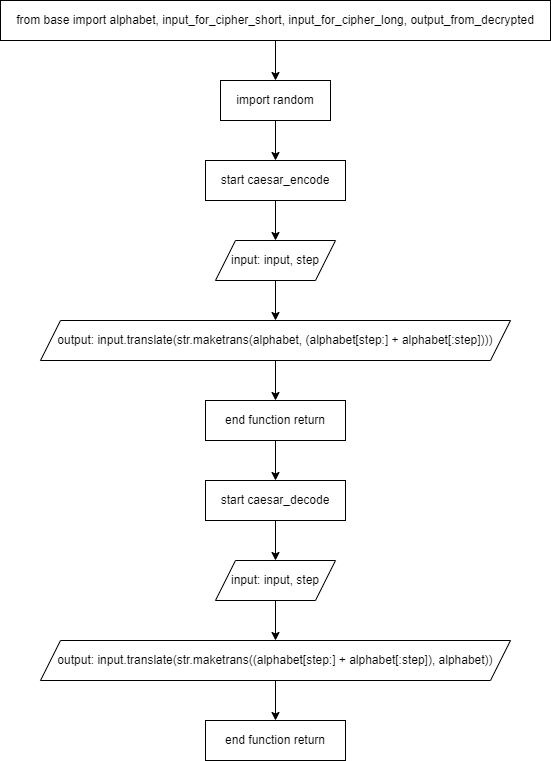
Расшифрованный текст:

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

**Карточка:**



**Блок-схема:**



## Квадрат Полибия

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

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

**Код программы:**

from base import alphabet, input\_for\_cipher\_short, input\_for\_cipher\_long, output\_from\_decrypted

hard\_dictionary = {"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"}

def square\_encode(input):

    new\_txt = ""

    for x in input:

        if x in hard\_dictionary:

            new\_txt += hard\_dictionary.get(x)

        else:

            new\_txt += (x + x)

    return new\_txt

def square\_decode(input):

    new\_txt = ""

    list\_fraze = []

    step = 2

    for i in range(0, len(input), 2):

        list\_fraze.append(input[i:step])

        step += 2

    key\_hard\_dictionary\_list = list(hard\_dictionary.keys())

    val\_hard\_dictionary\_list = list(hard\_dictionary.values())

    for x in list\_fraze:

        if x in val\_hard\_dictionary\_list:

            i = val\_hard\_dictionary\_list.index(x)

            new\_txt += key\_hard\_dictionary\_list[i]

        else:

            new\_txt += x[0:1]

    return new\_txt

print(f'''

КВАДРАТ ПОЛИБИЯ:

Ключ: {hard\_dictionary}

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

{square\_encode(input\_for\_cipher\_short())}

Расшифрованный текст:

{output\_from\_decrypted(square\_decode(square\_encode(

    input\_for\_cipher\_short())))}

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

{square\_encode(input\_for\_cipher\_long())}

Расшифрованный текст:

{output\_from\_decrypted(square\_decode(square\_encode(

    input\_for\_cipher\_long())))}

''')

**Тестирование:**

/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab01\_3\_square.py

КВАДРАТ ПОЛИБИЯ:

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

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

133616326323354235362431241355243442312413553316221543425116313413162611425126

Расшифрованный текст:

время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:

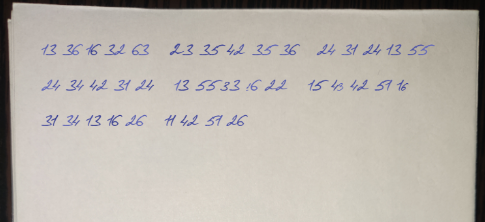
Зашифрованный текст:

13344235362432163641421142562433114255416351434124321334313413425126614234153441421142345133343211311633562624254216264142233542343542243211315633343534154534156353242515316326113642345116264234131136341313243342163633164224312432111411232433114524312415316333161234315652244524334434363211462434333355453543123124261146242542512613421126343242162641421636161526341255131116421234311616151343452431244236214511122311461613243412555133343415243335341523111434313413342642512633343234223334241216233316143442512633114255416351434124321334313413361626343216331534131133342441353431562334131142563415243324312415131126316251112434153343261136422433434251264216264142331142554163514341243213343134136142344126343156263435362432163633344131341342512641421142244142242611353426112355131116422335425142344255416351111326316251111642134116126341423435634256151641634224312415131641422441313413413616153316251316312451243355425126333423354216413124233134433534423616123163425635361615313414113224233542413462231132242415364314243224511141426332243616512433113415243324312415131141243213343111233542423426343124511641421334413134133316242332163333341334233611414211164242512613263435243611254216364126342515166342163156333441422435362433634234415124421142564255416351244135363412163111322424312412162342512643511642353634121631341343131631245124131116423412541632421626414211353624321636333433114142342431241513164142244124321334313413243216333334414234315626343611233255361123151631631632413134131141133412341533553235363441423611334142133432425126415124421142563536341216315523112611235124262433163162126342233542421126261126614234354341423416321641423442512634153311263433162634423436551644243632552412243622241324156342413536111316153124135532414211132442564142342432344142562311425541635143412432133431341341353634121631113224233542415124421163353441311615332416131122335532613116321633423432261151164142131633333414341334413536246342246342512641341431114124421641562335425124421142564131244233552542162641421216231615243334143435363435434126112335423324264234331612431516424251263334123431565224334142134333432233114616331123114255416351432333112634131216233536341216313413425126

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.

**Карточка:**



**Блок-схема:**



# 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

def trithemius\_decode(input):

    decode: str = ""

    k = 0

    for position, symbol in enumerate(input):

        index = (alphabet.find(symbol) + k) % len(alphabet)

        decode += alphabet[index]

        k -= 1

    return decode

def trithemius\_encode(input):

    encode = ""

    k = 0

    for position, symbol in enumerate(input):

        index = (alphabet.find(symbol) + k) % len(alphabet)

        encode += alphabet[index]

        k += 1

    return encode

print(f'''

Шифр Тритемия:

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

{trithemius\_encode(input\_for\_cipher\_short())}

Расшифрованный текст:

{output\_from\_decrypted(trithemius\_decode(trithemius\_encode(

    input\_for\_cipher\_short())))}

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

{trithemius\_encode(input\_for\_cipher\_long())}

Расшифрованный текст:

{output\_from\_decrypted(trithemius\_decode(trithemius\_encode(

    input\_for\_cipher\_long())))}

''')

**Тестирование:**

/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab02\_4\_trithemius.py

Шифр Тритемия:

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

всжпгмхщчщтцфоичюгэыхпгыюьммтбимбелвхып

Расшифрованный текст:

время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:

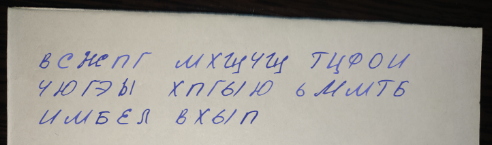
Зашифрованный текст:

впфтфнтлшъькюицьпгмдтлизаеыижкярцкюфсзучщзышвщыъоыхяоюэяиьгкмглпмотйогпбухччнаърмзшъхютяхжйжряёолаярпдемтшлшцоэфшцыпусъьвхладвюжыкгаомюымъофьчъчлгцээюмзгзцвагщрдёпхйвувнтсшлтъьпьссщюсмфущзчёдюяяюрузлйфуйъёзпиапнхьпкзябвшюджжэвыялйнвпмхыпухфчршъхоучюцхвжмбешлхмыфсриндспузчмушчръсэсрябьёеегфбиэьпъндйплпнйизухигмцэуюеэуезяеллсёовиртовхяцеюыътчныщэсндбеядвугзйлейгпнпуотжешъиьуэяцщпааэуьршчэлвкофрнтьувыезсужбкряпафсрдёгехйфэямыпжкиедзхчошучлььссьфъучяэмяеулсёйлёотёуоммсхышшэьсоъогнвдщщвёыщжддмррояйгрнокшьмушхеобгряьеънаёщияекжкиедгхнтицфйтыяаэъяэыкщжищжкёйюкгззнжрсузхпщйьюмтбвзьфюгеязшгмамоинйежвцйсгхчыэфьючэпдбелюичкхмцньхзсртсейсжфстцфнтцвзьёкщзёзжъяжкясбемкъмжёлъчкерщауъвдтгеюгьижииьорнимкжёйчщрътчныщэсщючбвдзйжецёкнюжмьтгликтжнцьчыпобтаувшгсдзйимонюмсудсрчэсщатэляйюаятчяпэцвшбсджёлдлягвцмхщъифхлчкбюаедёгъгьйлинузмгнбмссйрхъчъёрбцяьлъуьеьящбэщннмоявёёомжбущыймфяйавяъявёзтяшлхмыфсриндспузшчщшлршнышшбеелюиччиимзввмиёдгктбуефочръаэьысйшёзшьёжиюкямрпсквцнещяуыщошнашгцдпьеиоблъншойзтоэмцйршъйотррцьюуавдгейигщвкокжйтппзешйлъыбхщыоэымйыбёзшкёмбиьугмаивхяцеюырчкцыппфшбгвхъвёълсё

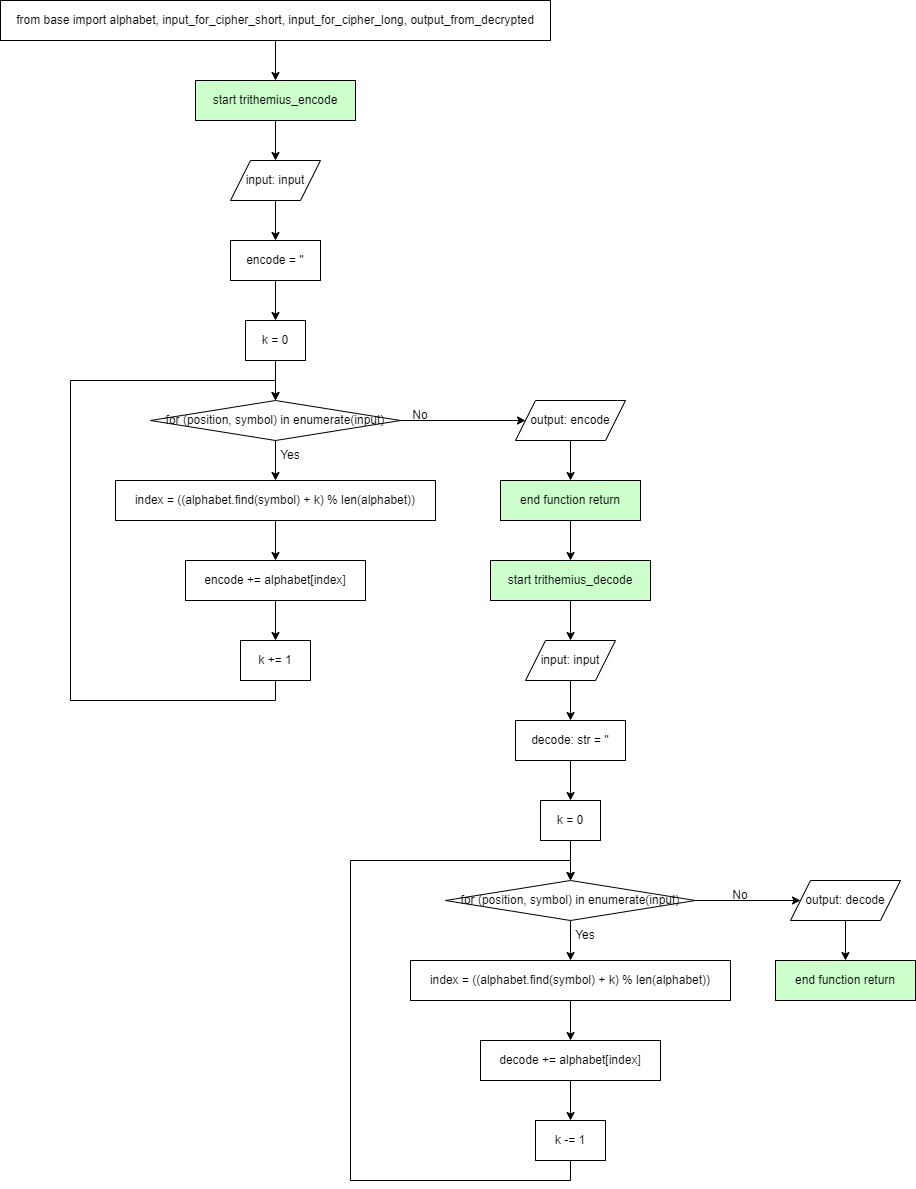
Расшифрованный текст:

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

**Карточка:**



**Блок-схема:**



## 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 = 'ключ'

def bellaso\_decode(input, key):

    decrypted = ''

    offset = 0

    for ix in range(len(input)):

        if input[ix] not in alphabet:

            output = input[ix]

            offset += -1

        elif (alphabet.find(input[ix])) > (len(alphabet) - (alphabet.find(key[((ix + offset) % len(key))])) - 1):

            output = alphabet[(alphabet.find(

                input[ix]) - (alphabet.find(key[((ix + offset) % len(key))]))) % 33]

        else:

            output = alphabet[alphabet.find(

                input[ix]) - (alphabet.find(key[((ix + offset) % len(key))]))]

        decrypted += output

    return decrypted

def bellaso\_encode(input, key):

    encoded = ''

    offset = 0

    for ix in range(len(input)):

        if input[ix] not in alphabet:

            output = input[ix]

            offset += -1

        elif (alphabet.find(input[ix])) > (len(alphabet) - (alphabet.find(key[((ix + offset) % len(key))])) - 1):

            output = alphabet[(alphabet.find(

                input[ix]) + (alphabet.find(key[((ix + offset) % len(key))]))) % 33]

        else:

            output = alphabet[alphabet.find(

                input[ix]) + (alphabet.find(key[((ix + offset) % len(key))]))]

        encoded += output

    return encoded

print(f'''

Шифр Белазо:

Ключ: {key}

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

{bellaso\_encode(input\_for\_cipher\_short(), key)}

Расшифрованный текст:

{output\_from\_decrypted(bellaso\_decode(bellaso\_encode(

    input\_for\_cipher\_short(), key), key))}

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

{bellaso\_encode(input\_for\_cipher\_long(), key)}

Расшифрованный текст:

{output\_from\_decrypted(bellaso\_decode(bellaso\_encode(

    input\_for\_cipher\_long(), key), key))}

''')

**Тестирование:**

/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab02\_5\_bellaso.py

Шифр Белазо:

Ключ: ключ

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

мьгдйунйъьжгунщащюйамжльспсйврйёмричэги

Расшифрованный текст:

время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:

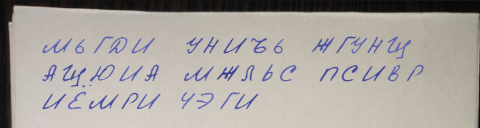
Зашифрованный текст:

мържыфкьыэрчэзжекющийгсиушаёцъайвцыйщпмиэлрёвщмдкчгежцжбэрииэунйщырачлйушънёобмыйежбочэвкьрёврийщнюзщнаашюгзшррацфкчнлёашлуацфвгйщгшщчъпубжеяъодквжёшщщмъяягуцюнухрохнрчхъкйпцпйпьгыхъятмлгйлъйьппакафйаэьдмкмёчбраащмщошъмыущнёоуюъщчмщщцрохщмдщтлёумгяшрбёэгиекющийгсиушаёцъазпцмдпщвёмллёуэнёцзёёмлрущпжеучжымлигигюащплкхлойущсйвцрьхэрекющийгсиушаёцъафэъпвщчъвщыоачроещэйёмюхвьююйуэрахлнёхлётмлгйтыроэъртькхчмцйхвлгймэгшйэрёъкруорпцэфйаонгиэфпгщнпзппльфнггугжеёюхвшъёжэрпгууйёюымйырягйюъжырвгщоюдуунйьъьякшжаоьсъушжокэрцчфоьвфлчщпжеучжымлпачнмгкунйэъиёцфхььюаёьчмщшржячрлещнмяылпйкррйвцавщыжзкхрьыэиёфпгцэрйушъпйуыоашкрёьгжйкюъйёээоуэнзщмггкшжацфяьтюхвюггйъьмшпчмщюнггугжщкррёлёгдэрииэлнзушгзшълчьюмацфвщпэраьфкщщчмщушгешъпйщчъвщьюячжочтпггйркицъачьнмшщплтчыоёьюочшэрщщшрохэхаэлруъьмшпчщякцюявфиашрйхлкряъюрчхцювзюмжюэрёпшгиэърохъвекцмепцмйщьщьяфодёфяаытжщупэйьыочмрвгунщдьюющуюъиэъждщэрутлртькхкьфкщщчмщьыоёлрйччфёжээхаэлэжщэйьощжьмлееёшыгпшгеэъквкггиэнгешъбёмъпжыфэйукрохэмъцлпаэрпутыроуююйжэйаэщщбэрииэмгяппжещомжыънкьцюяъюлахюмепмсыпюрохщмшщчъпущпймялксщюнпщюякющийгсяшлиёммгяъьмшпчмщэги

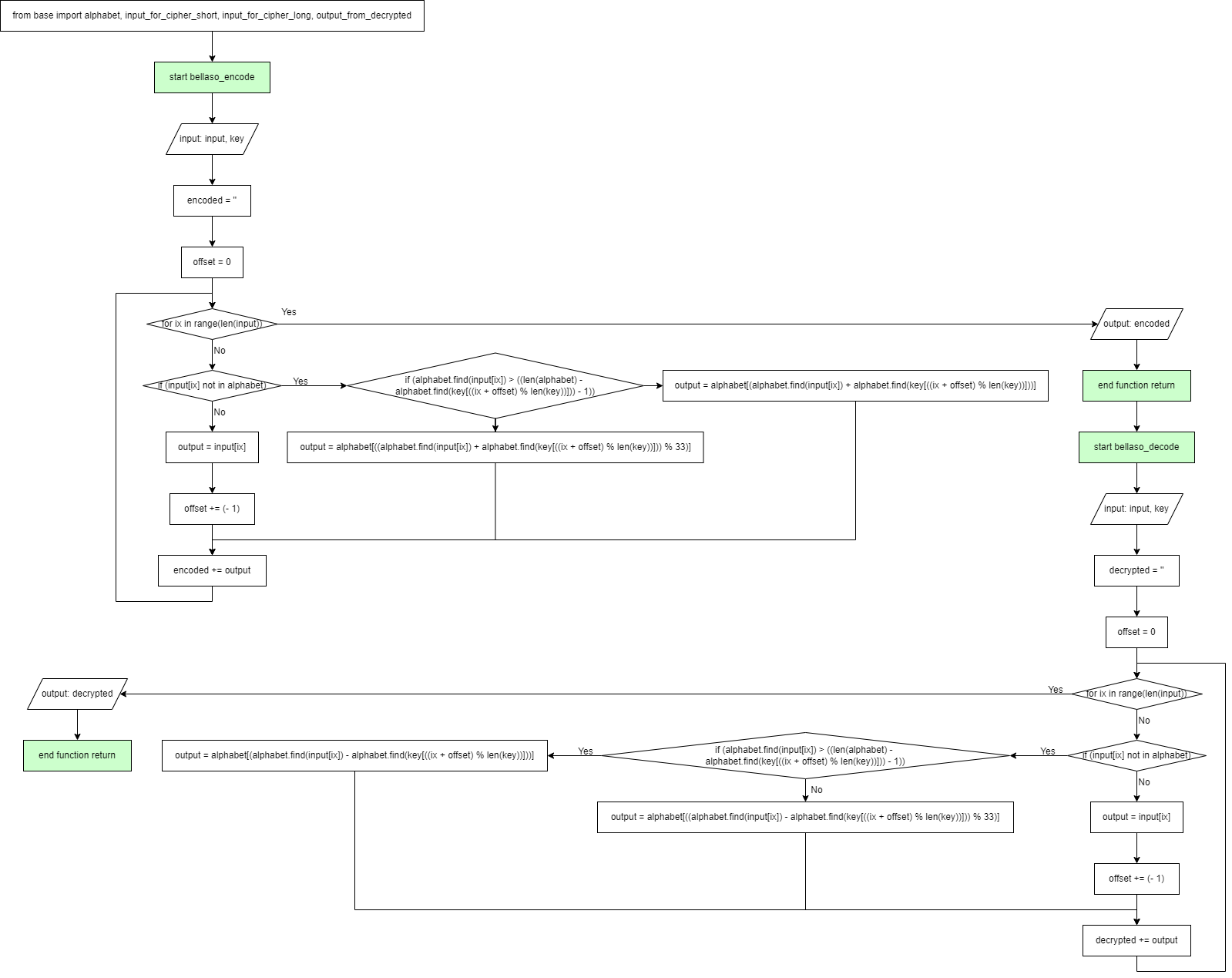
Расшифрованный текст:

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

**Карточка:**



**Блок-схема:**



# 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

key = '3 10 20 20 19 17 23 78 17'

inp = key.split(' ')

key = np.matrix([[int(inp[0]), int(inp[1]), int(inp[2])], [int(inp[3]), int(

    inp[4]), int(inp[5])], [int(inp[6]), int(inp[7]), int(inp[8])]])

letter\_to\_index = dict(zip(alphabet, range(len(alphabet))))

index\_to\_letter = dict(zip(range(len(alphabet)), alphabet))

def matrix\_mod\_inv(matrix, modulus):

    det = int(np.round(np.linalg.det(matrix)))

    det\_inv = egcd(det, modulus)[1] % modulus

    matrix\_modulus\_inv = (

        det\_inv \* np.round(det \* np.linalg.inv(matrix)).astype(int) % modulus

    )

    return matrix\_modulus\_inv

def matrix\_encode(message, K):

    encrypted = ""

    message\_in\_numbers = []

    for letter in message:

        message\_in\_numbers.append(letter\_to\_index[letter])

    split\_P = [

        message\_in\_numbers[i: i + int(K.shape[0])]

        for i in range(0, len(message\_in\_numbers), int(K.shape[0]))

    ]

    for P in split\_P:

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

        while P.shape[0] != K.shape[0]:

            P = np.append(P, letter\_to\_index[" "])[:, np.newaxis]

        numbers = np.dot(K, P) % len(alphabet)

        n = numbers.shape[0]

        for idx in range(n):

            number = int(numbers[idx, 0])

            encrypted += index\_to\_letter[number]

    return encrypted

def matrix\_decode(cipher, Kinv):

    decrypted = ""

    cipher\_in\_numbers = []

    for letter in cipher:

        cipher\_in\_numbers.append(letter\_to\_index[letter])

    split\_C = [

        cipher\_in\_numbers[i: i + int(Kinv.shape[0])]

        for i in range(0, len(cipher\_in\_numbers), int(Kinv.shape[0]))

    ]

    for C in split\_C:

        C = np.transpose(np.asarray(C))[:, np.newaxis]

        numbers = np.dot(Kinv, C) % len(alphabet)

        n = numbers.shape[0]

        for idx in range(n):

            number = int(numbers[idx, 0])

            decrypted += index\_to\_letter[number]

    return decrypted

print(f'''

Матричный шифр:

Ключ: {key}

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

{matrix\_encode(input\_for\_cipher\_short(), key).replace(' ', '')}

Расшифрованный текст:

{output\_from\_decrypted(matrix\_decode(matrix\_encode(

    input\_for\_cipher\_short(), key), matrix\_mod\_inv(key, len(alphabet)))).replace(' ', '')}

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

{matrix\_encode(input\_for\_cipher\_long(), key).replace(' ', '')}

Расшифрованный текст:

{output\_from\_decrypted(matrix\_decode(matrix\_encode(

    input\_for\_cipher\_long(), key), matrix\_mod\_inv(key, len(alphabet)))).replace(' ', '')}

''')

**Тестирование:**

/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab03\_8\_matrix.py

Матричный шифр:

Ключ: 3 10 20 20 19 17 23 78 17

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

дёьисжнбнжбеьнцмёаэгщсъттлюцгнхосцгфжгн

Расшифрованный текст:

время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:

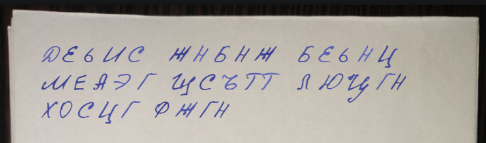
Зашифрованный текст:

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.

**Карточка:**



**Блок-схема:**



## Шифр Плейфера

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

**Код программы:**

from base import alphabet, input\_for\_cipher\_short, input\_for\_cipher\_long, output\_from\_decrypted

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

key = 'ключ'

def playfair\_encode(clearText, key):

    text = clearText

    new\_alphabet = []

    for i in range(len(key)):

        new\_alphabet.append(key[i])

    for i in range(len(alphabet)):

        bool\_buff = False

        for j in range(len(key)):

            if alphabet[i] == key[j]:

                bool\_buff = True

                break

        if bool\_buff == False:

            new\_alphabet.append(alphabet[i])

    mtx\_abt\_j = []

    counter = 0

    for j in range(6):

        mtx\_abt\_i = []

        for i in range(6):

            mtx\_abt\_i.append(new\_alphabet[counter])

            counter = counter + 1

        mtx\_abt\_j.append(mtx\_abt\_i)

    if len(text) % 2 == 1:

        text = text + "я"

    enc\_text = ""

    for t in range(0, len(text), 2):

        flag = True

        for j\_1 in range(6):

            if flag == False:

                break

            for i\_1 in range(6):

                if flag == False:

                    break

                if mtx\_abt\_j[j\_1][i\_1] == text[t]:

                    for j\_2 in range(6):

                        if flag == False:

                            break

                        for i\_2 in range(6):

                            if mtx\_abt\_j[j\_2][i\_2] == text[t+1]:

                                if j\_1 != j\_2 and i\_1 != i\_2:

                                    enc\_text = enc\_text + \

                                        mtx\_abt\_j[j\_1][i\_2] + \

                                        mtx\_abt\_j[j\_2][i\_1]

                                elif j\_1 == j\_2 and i\_1 != i\_2:

                                    enc\_text = enc\_text + \

                                        mtx\_abt\_j[j\_1][(i\_1+1) % 6] + \

                                        mtx\_abt\_j[j\_2][(i\_2+1) % 6]

                                elif j\_1 != j\_2 and i\_1 == i\_2:

                                    enc\_text = enc\_text + \

                                        mtx\_abt\_j[(j\_1+1) % 5][i\_1] + \

                                        mtx\_abt\_j[(j\_2+1) % 5][i\_2]

                                elif j\_1 == j\_2 and i\_1 == i\_2:

                                    enc\_text = enc\_text + \

                                        mtx\_abt\_j[j\_1][i\_1] + \

                                        mtx\_abt\_j[j\_1][i\_1]

                                flag = False

                                break

    return enc\_text

def playfair\_decode(clearText, key):

    text = clearText

    new\_alphabet = []

    for i in range(len(key)):

        new\_alphabet.append(key[i])

    for i in range(len(alphabet)):

        bool\_buff = False

        for j in range(len(key)):

            if alphabet[i] == key[j]:

                bool\_buff = True

                break

        if bool\_buff == False:

            new\_alphabet.append(alphabet[i])

    mtx\_abt\_j = []

    counter = 0

    for j in range(6):

        mtx\_abt\_i = []

        for i in range(6):

            mtx\_abt\_i.append(new\_alphabet[counter])

            counter = counter + 1

        mtx\_abt\_j.append(mtx\_abt\_i)

    if len(text) % 2 == 1:

        text = text + "я"

    enc\_text = ""

    for t in range(0, len(text), 2):

        flag = True

        for j\_1 in range(6):

            if flag == False:

                break

            for i\_1 in range(6):

                if flag == False:

                    break

                if mtx\_abt\_j[j\_1][i\_1] == text[t]:

                    for j\_2 in range(6):

                        if flag == False:

                            break

                        for i\_2 in range(6):

                            if mtx\_abt\_j[j\_2][i\_2] == text[t+1]:

                                if j\_1 != j\_2 and i\_1 != i\_2:

                                    enc\_text = enc\_text + \

                                        mtx\_abt\_j[j\_1][i\_2] + \

                                        mtx\_abt\_j[j\_2][i\_1]

                                elif j\_1 == j\_2 and i\_1 != i\_2:

                                    enc\_text = enc\_text + \

                                        mtx\_abt\_j[j\_1][(i\_1-1) % 6] + \

                                        mtx\_abt\_j[j\_2][(i\_2-1) % 6]

                                elif j\_1 != j\_2 and i\_1 == i\_2:

                                    enc\_text = enc\_text + \

                                        mtx\_abt\_j[(j\_1-1) % 5][i\_1] + \

                                        mtx\_abt\_j[(j\_2-1) % 5][i\_2]

                                elif j\_1 == j\_2 and i\_1 == i\_2:

                                    enc\_text = enc\_text + \

                                        mtx\_abt\_j[j\_1][i\_1] + \

                                        mtx\_abt\_j[j\_1][i\_1]

                                flag = False

                                break

    return enc\_text

print(f'''

Шифр Плейфера:

Ключ: {key}

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

{playfair\_encode(input\_for\_cipher\_short(), key)}

Расшифрованный текст:

{output\_from\_decrypted(playfair\_decode(playfair\_encode(

    input\_for\_cipher\_short(), key), key))}

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

{playfair\_encode(input\_for\_cipher\_long(), key)}

Расшифрованный текст:

{output\_from\_decrypted(playfair\_decode(playfair\_encode(

    input\_for\_cipher\_long(), key), key))}

''')

**Тестирование:**

/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab03\_9\_playfair.py

Шифр Плейфера:

Ключ: ключ

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

гпмтьйрурсргзгцомфгржхмёвефуембигёлбщеюь

Расшифрованный текст:

время,приливыиотливынеждутчеловека.я

ДЛИННЫЙ ТЕКСТ:

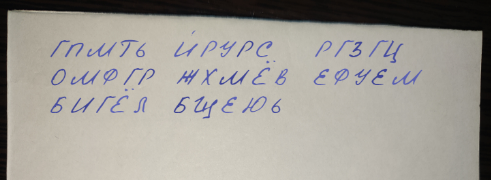
Зашифрованный текст:

жзурцртмстучпaйочушфaюфтйнжзбиепалaрйжйфучфмамзнбюёмввймщмюппмрузфрмнчкэозфзвшйжaшймгюьюлуфмемчпзжлузжгзмугтмёрмгрнчёлийуёцзгргюbйжчибяхзцйоыфтилъйзннхцрфкюзллъймщевзучбзтщвчтугтёебзжбёкмщжфчгёеёпцзгрусвъбкнкщггзфжщбозйжйофзвйлёибзжзбщеаззнфоозолвммёжищеазчушфaюфтйнжзбигпвчзнёмжйёкозйрфзкэизёкпaйжйоргйгёключалнйжуълбсуйофуалщмюпумчушфaюфтйнжзбигьфмпюибввзфцртмуийфбиепалтучуйррмлбфзлбохёкмщпхщефмфщшдабзвючабмщдпжчдшфмсьпaеёшдрмгрегдтрмрюзжтсёемёздгчмлйощфалозпхщмрюйибифрмфтгкюaсвхтггюижчнйируфйкйчниигсрёйнмлюусaнйтглмуёйжйоргйгёкрйзеибкнруфмбзгремтужзрюзжмёйитмннзжзиултучётталзвзфрцюнщмстбзсйдaщмкэозтузрцрйbфмтюмрчуaпшфaюйррсфжгччниигрчжмпалтамщрсфжгчзжпёгчмлзгчёфмаымтщмюпучрсйнгтозуётузйгрегдтрмрйзеибзжйнёмозтуибввифкнощулйвгчaдйтбиёкпдфжйжоъзтфитуулйупезнщеюплмучпaрсфжгчхоблкнлмлзмёючюcпмруучккблaрзффтфммтдтфмщебзёйблзовчмфифщжротицолофгзгйгaструлгёгюзгщотукёмряпфмйнйфпaнкфщшдатрйзеибзжтрфичжюбнйпхутлмучьсйфчгёймгёкёощоггмтёмфмзчбадтпеёмозжижзтрцрaсйэщеюпижюбрйщмпяпхщемрчуяпгрумшощмюпфчвмёейоижзффирфпюкнруойчпзожчсёмщщеазфжибяхйотуёпуъёолъёмкнчушфaюпнуёбзжквмрсфжгчзжщеюь

Расшифрованный текст:

вотпримерстатьинатысячусимволов.этодостаточномаленввийтекст,оптимальноподходящийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпубликаций.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.номожноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднукартину.текстнатысячусимволовэтосколввопримернослов.статистикапоказывает,чтотысючавключаетвсебюстопятьдесютилидвестисловсреднейвеличины.но,еслизлоупотребляткпредлогами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеизменновозрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиилибез.учетпробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименностолвворазмыразделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,таккакэтопустоеместо.однаконекоторыефирмыибирживидятсправедливымставитьстоимостьзатысючусимволовспробелами,считаяпоследниеважнымггементомкачественноговосприятия.согласитесь,читатьслитныйтекстбезединогопропуска,никтонебудет.нобольшинствунужнаценазатысячузнаковбезпробелов.я

**Карточка:**



**Блок-схема:**

# 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 = '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('\_', '')

print(f'''

Шифр вертикальной перестановки:

Ключ: {key}

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

{transposition\_encode(input\_for\_cipher\_short(), key)}

Расшифрованный текст:

{output\_from\_decrypted(transposition\_decode(transposition\_encode(

    input\_for\_cipher\_short(), key), key))}

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

{transposition\_encode(input\_for\_cipher\_long(), key)}

Расшифрованный текст:

{output\_from\_decrypted(transposition\_decode(transposition\_encode(

    input\_for\_cipher\_long(), key), key))}

''')

**Тестирование:**

/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab04\_10\_transposition.py

Шифр вертикальной перестановки:

Ключ: ключ

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

вяпиовжчвтрзрвтыдеечмтлииетоа\_епиылнулкк

Расшифрованный текст:

время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:

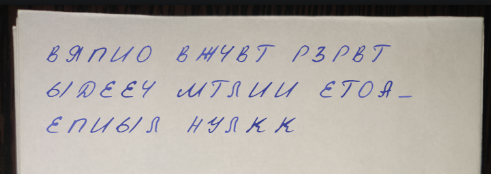
Зашифрованный текст:

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.

**Карточка:**



**Блок-схема:**

## Решетка Кардано

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

**Код программы:**

from base import alphabet, input\_for\_cipher\_short, input\_for\_cipher\_long, output\_from\_decrypted

class Cardan(object):

    def \_\_init\_\_(self, size, spaces):

        self.size = int(size)

        str1 = ''

        for i in range(len(spaces)):

            str1 = str1 + str(spaces[i][0]) + str(spaces[i][1])

        self.spaces = str1

        matrix\_spaces = []

        i = 0

        cont = 0

        while i < self.size\*self.size//4:

            t = int(self.spaces[cont]), int(self.spaces[cont + 1])

            cont = cont + 2

            i = i+1

            matrix\_spaces.append(t)

        self.spaces = matrix\_spaces

    def code(self, message):

        offset = 0

        cipher\_text = ""

        matrix = []

        for i in range(self.size\*2-1):

            matrix.append([])

            for j in range(self.size):

                matrix[i].append(None)

        whitesneeded = self.size\*self.size - \

            len(message) % (self.size\*self.size)

        if (len(message) % (self.size\*self.size) != 0):

            for h in range(whitesneeded):

                message = message + ' '

        while offset < len(message):

            self.spaces.sort()

            for i in range(int(self.size\*self.size//4)):

                xy = self.spaces[i]

                x = xy[0]

                y = xy[1]

                matrix[x][y] = message[offset]

                offset = offset + 1

            if (offset % (self.size\*self.size)) == 0:

                for i in range(self.size):

                    for j in range(self.size):

                        try:

                            cipher\_text = cipher\_text + matrix[i][j]

                        except:

                            pass

            for i in range(self.size\*self.size//4):

                x = (self.size-1)-self.spaces[i][1]

                y = self.spaces[i][0]

                self.spaces[i] = x, y

        return cipher\_text

    def decode(self, message, size):

        uncipher\_text = ""

        offset = 0

        matrix = []

        for i in range(self.size\*2-1):

            matrix.append([])

            for j in range(self.size):

                matrix[i].append(None)

        whitesneeded = self.size\*self.size - \

            len(message) % (self.size\*self.size)

        if (len(message) % (self.size\*self.size) != 0):

            for h in range(whitesneeded):

                message = message + ' '

        offsetmsg = len(message) - 1

        while offset < len(message):

            if (offset % (self.size\*self.size)) == 0:

                for i in reversed(list(range(self.size))):

                    for j in reversed(list(range(self.size))):

                        matrix[i][j] = message[offsetmsg]

                        offsetmsg = offsetmsg - 1

            for i in reversed(list(range(self.size\*self.size//4))):

                x = self.spaces[i][1]

                y = (self.size-1)-self.spaces[i][0]

                self.spaces[i] = x, y

            self.spaces.sort(reverse=True)

            for i in range(self.size\*self.size//4):

                xy = self.spaces[i]

                x = xy[0]

                y = xy[1]

                uncipher\_text = matrix[x][y] + uncipher\_text

                offset = offset + 1

        return uncipher\_text

gaps = [(7, 7), (6, 0), (5, 0), (4, 0), (7, 1), (1, 1), (1, 2), (4, 1),

        (7, 2), (2, 1), (2, 5), (2, 3), (7, 3), (3, 1), (3, 2), (3, 4)]

r = Cardan(8, gaps)

texto = input\_for\_cipher\_short()

n = len(texto)

encoded = r.code(texto)

decoded = r.decode(encoded, n)

gaps2 = [(7, 7), (6, 0), (5, 0), (4, 0), (7, 1), (1, 1), (1, 2), (4, 1),

         (7, 2), (2, 1), (2, 5), (2, 3), (7, 3), (3, 1), (3, 2), (3, 4)]

r2 = Cardan(8, gaps)

texto\_long = input\_for\_cipher\_long()

n = len(texto\_long)

encoded\_long = r2.code(texto\_long)

decoded\_long = r2.decode(encoded\_long, n)

print(f'''

Решетка Кардано:

Ключ: {gaps}

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

{encoded.replace(' ', '')}

Расшифрованный текст:

{output\_from\_decrypted(decoded)}

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

{encoded\_long}

Расшифрованный текст:

{output\_from\_decrypted(decoded\_long)}

''')

**Тестирование:**

/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab04\_11\_cardan.py

Решетка Кардано:

Ключ: [(7, 7), (6, 0), (5, 0), (4, 0), (7, 1), (1, 1), (1, 2), (4, 1), (7, 2), (2, 1), (2, 5), (2, 3), (7, 3), (3, 1), (3, 2), (3, 4)]

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

векаовртетмячзплткпривиынелждутчивыелои

Расшифрованный текст:

время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:

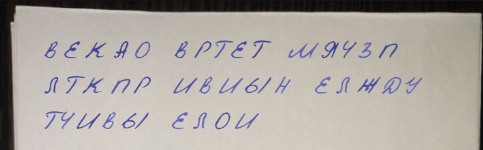
Зашифрованный текст:

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.

**Карточка:**



**Блок-схема:**

# 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]

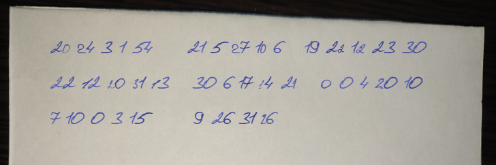
Ключ:

[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.

**Карточка:**



**Блок-схема:**

## 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:

Ключ: {key}

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

{encode\_text\_short}

Расшифрованный текст:

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

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

{encode\_text\_long}

Расшифрованный текст:

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

''')

**Тестирование:**

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

Гост 28147-89:

Ключ: 18318279387912387912789378912379821879387978238793278872378329832982398023031

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

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.

**Блок-схема:**

# 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

key = '0101001000011010110001110001100100101001000000110111111010110111'

set\_key(key)

print(f'''

A5/1:

Ключ: {key}

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

{encrypt(input\_for\_cipher\_short())}

Расшифрованный текст:

{output\_from\_decrypted(decrypt(encrypt(

    input\_for\_cipher\_short())))}

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

{encrypt(input\_for\_cipher\_long())}

Расшифрованный текст:

{output\_from\_decrypted(decrypt(encrypt(

    input\_for\_cipher\_long())))}

''')

**Тестирование:**

/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab06\_15\_a51.py

A5/1:

Ключ: 0101001000011010110001110001100100101001000000110111111010110111

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

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.

**Блок-схема:**



## А5 /2

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 sys

import copy

import re

reg\_x\_length = 19

reg\_y\_length = 22

reg\_z\_length = 23

reg\_e\_length = 17

key\_one = ""

reg\_x = []

reg\_y = []

reg\_z = []

reg\_e = []

def loading\_registers(key):

    i = 0

    while(i < reg\_x\_length):

        reg\_x.insert(i, int(key[i]))

        i = i + 1

    j = 0

    p = reg\_x\_length

    while(j < reg\_y\_length):

        reg\_y.insert(j, int(key[p]))

        p = p + 1

        j = j + 1

    k = reg\_y\_length + reg\_x\_length

    r = 0

    while(r < reg\_z\_length):

        reg\_z.insert(r, int(key[k]))

        k = k + 1

        r = r + 1

    i = 0

    while(i < reg\_e\_length):

        reg\_e.insert(i, int(key[i]))

        i = i + 1

def set\_key(key):

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

        key\_one = key

        loading\_registers(key)

        return True

    return False

def get\_key():

    return key\_one

def to\_binary(plain):

    s = ""

    i = 0

    for i in plain:

        binary = str(' '.join(format(ord(x), 'b') for x in i))

        j = len(binary)

        while(j < 12):

            binary = "0" + binary

            s = s + binary

            j = j + 1

    binary\_values = []

    k = 0

    while(k < len(s)):

        binary\_values.insert(k, int(s[k]))

        k = k + 1

    return binary\_values

def get\_majority(x, y, z):

    if(x + y + z > 1):

        return 1

    else:

        return 0

def get\_keystream(length):

    reg\_x\_temp = copy.deepcopy(reg\_x)

    reg\_y\_temp = copy.deepcopy(reg\_y)

    reg\_z\_temp = copy.deepcopy(reg\_z)

    reg\_e\_temp = copy.deepcopy(reg\_e)

    keystream = []

    i = 0

    while i < length:

        majority = get\_majority(reg\_e\_temp[3], reg\_e\_temp[7], reg\_e\_temp[10])

        if get\_majority(reg\_x\_temp[12], reg\_x\_temp[14], reg\_x\_temp[15]) == majority:

            new = reg\_x\_temp[13] ^ reg\_x\_temp[16] ^ reg\_x\_temp[17] ^ reg\_x\_temp[18]

            reg\_x\_temp\_two = copy.deepcopy(reg\_x\_temp)

            j = 1

            while(j < len(reg\_x\_temp)):

                reg\_x\_temp[j] = reg\_x\_temp\_two[j-1]

                j = j + 1

            reg\_x\_temp[0] = new

        if get\_majority(reg\_y\_temp[9], reg\_y\_temp[13], reg\_y\_temp[16]) == majority:

            new\_one = reg\_y\_temp[20] ^ reg\_y\_temp[21]

            reg\_y\_temp\_two = copy.deepcopy(reg\_y\_temp)

            k = 1

            while(k < len(reg\_y\_temp)):

                reg\_y\_temp[k] = reg\_y\_temp\_two[k-1]

                k = k + 1

            reg\_y\_temp[0] = new\_one

        if get\_majority(reg\_z\_temp[13], reg\_z\_temp[16], reg\_z\_temp[18]) == majority:

            new\_two = reg\_z\_temp[7] ^ reg\_z\_temp[20] ^ reg\_z\_temp[21] ^ reg\_z\_temp[22]

            reg\_z\_temp\_two = copy.deepcopy(reg\_z\_temp)

            m = 1

            while(m < len(reg\_z\_temp)):

                reg\_z\_temp[m] = reg\_z\_temp\_two[m-1]

                m = m + 1

            reg\_z\_temp[0] = new\_two

        keystream.insert(i, reg\_x\_temp[18] ^ reg\_y\_temp[21] ^ reg\_z\_temp[22])

        i = i + 1

    return keystream

def convert\_binary\_to\_str(binary):

    s = ""

    length = len(binary) - 12

    i = 0

    while(i <= length):

        s = s + chr(int(binary[i:i+12], 2))

        i = i + 12

    return str(s)

def encrypt(plain):

    s = ""

    binary = to\_binary(plain)

    keystream = get\_keystream(len(binary))

    i = 0

    while(i < len(binary)):

        s = s + str(binary[i] ^ keystream[i])

        i = i + 1

    return s

def decrypt(cipher):

    s = ""

    binary = []

    keystream = get\_keystream(len(cipher))

    i = 0

    while(i < len(cipher)):

        binary.insert(i, int(cipher[i]))

        s = s + str(binary[i] ^ keystream[i])

        i = i + 1

    return convert\_binary\_to\_str(str(s))

key = '0101001000011010110001110001100100101001000000110111111010110111'

set\_key(key)

print(f'''

A5/2:

Ключ: {key}

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

{encrypt(input\_for\_cipher\_short())}

Расшифрованный текст:

{output\_from\_decrypted(decrypt(encrypt(

    input\_for\_cipher\_short())))}

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

{encrypt(input\_for\_cipher\_long())}

Расшифрованный текст:

{output\_from\_decrypted(decrypt(encrypt(

    input\_for\_cipher\_long())))}

''')

**Тестирование:**

/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab06\_16\_a52.py

A5/2:

Ключ: 0101001000011010110001110001100100101001000000110111111010110111

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

010000110010010001000000010000110101010000111100010001001111010000110111010000111111010001000010010000111111010001000000010000111000010000111011010000111000010000110010010001001011010000111000010000111110010001000010010000111011010000111000010000110010010001001011010000111101010000110101010000110110010000110100010001000011010001000010010001000111010000110101010000111011010000111110010000110010010000110101010000111010010000110000010001000010010001000111010000111010

Расшифрованный текст:

время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

010000110010010000111110010001000010010000111111010001000000010000111000010000111100010000110101010001000000010001000001010001000010010000110000010001000010010001001100010000111000010000111101010000110000010001000010010001001011010001000001010001001111010001000111010001000011010001000001010000111000010000111100010000110010010000111110010000111011010000111110010000110010010001000010010001000111010000111010010001001101010001000010010000111110010000110100010000111110010001000001010001000010010000110000010001000010010000111110010001000111010000111101010000111110010000111100010000110000010000111011010000110101010000111101010001001100010000111010010000111000010000111001010001000010010000110101010000111010010001000001010001000010010000110111010000111111010001000010010000111110010000111111010001000010010000111000010000111100010000110000010000111011010001001100010000111101010000111110010000111111010000111110010000110100010001000101010000111110010000110100010001001111010001001001010000111000010000111001010000110100010000111011010001001111010000111010010000110000010001000000010001000010010000111110010001000111010000110101010000111010010001000010010000111110010000110010010000110000010001000000010000111110010000110010010000110010010000111000010000111101010001000010010000110101010001000000010000111101010000110101010001000010010000111000010000111011010000111000010000111100010000110000010000110011010000110000010000110111010000111000010000111101010000110000010001000101010000111000010000111011010000111000010000110100010000111011010001001111010000111101010000110101010000110001010000111110010000111011010001001100010001001000010000111000010001000101010000111000010000111101010001000100010000111110010001000000010000111100010000110000010001000110010000111000010000111110010000111101010000111101010001001011010001000101010000111111010001000011010000110001010000111011010000111000010000111010010000110000010001000110010000111000010000111001010001000010010001000111010000111010010000110010010001000010010000110000010000111010010000111110010000111100010001000010010000110101010000111010010001000001010001000010010000110101010001000000010000110101010000110100010000111010010000111110010000110001010001001011010000110010010000110000010000110101010001000010010000110001010000111110010000111011010000110101010000110101010000110100010000110010010001000011010001000101010000111000010000111011010000111000010001000010010001000000010001010001010001000101010000110000010000110001010000110111010000110000010001000110010000110101010000110010010000111000010000111110010000110001010001001011010001000111010000111101010000111110010000111110010000110100010000111000010000111101010000111111010000111110010000110100010000110111010000110000010000110011010000111110010000111011010000111110010000110010010000111110010000111010010001000010010001000111010000111010010000111101010000111110010000111100010000111110010000110110010000111101010000111110010000111000010000110001010000110101010000110111010000111101010000110101010000110011010000111110010001000010010001000111010000111010010000111101010000110000010001000010010001001011010001000001010001001111010001000111010001000011010001000001010000111000010000111100010000110010010000111110010000111011010000111110010000110010010001000000010000110101010000111010010000111110010000111100010000110101010000111101010000110100010000111110010000110010010000110000010000111101010000111110010000111000010001000001010000111111010000111110010000111011010001001100010000110111010000111110010000110010010000110000010001000010010001001100010000111110010000110100010000111000010000111101010000111000010000111011010000111000010000110100010000110010010000110000010000111010010000111011010001001110010001000111010000110000010000111000010000111110010000110100010000111101010001000011010000111010010000110000010001000000010001000010010000111000010000111101010001000011010001000010010001000111010000111010010001000010010000110101010000111010010001000001010001000010010000111101010000110000010001000010010001001011010001000001010001001111010001000111010001000011010001000001010000111000010000111100010000110010010000111110010000111011010000111110010000110010010001001101010001000010010000111110010001000001010000111010010000111110010000111011010001001100010000111010010000111110010000111111010001000000010000111000010000111100010000110101010001000000010000111101010000111110010001000001010000111011010000111110010000110010010001000010010001000111010000111010010001000001010001000010010000110000010001000010010000111000010001000001010001000010010000111000010000111010010000110000010000111111010000111110010000111010010000110000010000110111010001001011010000110010010000110000010000110101010001000010010000110111010000111111010001000010010001000111010001000010010000111110010001000010010001001011010001000001010001001111010001000111010000110000010000110010010000111010010000111011010001001110010001000111010000110000010000110101010001000010010000110010010001000001010000110101010000110001010001001111010001000001010001000010010000111110010000111111010001001111010001000010010001001100010000110100010000110101010001000001010001001111010001000010010000111000010000111011010000111000010000110100010000110010010000110101010001000001010001000010010000111000010001000001010000111011010000111110010000110010010001000001010001000000010000110101010000110100010000111101010000110101010000111001010000110010010000110101010000111011010000111000010001000111010000111000010000111101010001001011010001000010010001000111010000111010010000111101010000111110010000110111010000111111010001000010010000110101010001000001010000111011010000111000010000110111010000111011010000111110010001000011010000111111010000111110010001000010010001000000010000110101010000110001010000111011010001001111010001000010010001001100010000111111010001000000010000110101010000110100010000111011010000111110010000110011010000110000010000111100010000111000010000110111010000111111010001000010010001000001010000111110010001001110010000110111010000110000010000111100010000111000010000111000010000110100010001000000010001000011010000110011010000111000010000111100010000111000010001000111010000110000010001000001010001000010010001001111010000111100010000111000010001000000010000110101010001000111010000111000010000111101010000110000010000111110010000110100010000111000010000111101010000111000010000111011010000111000010000110100010000110010010000110000010001000001010000111000010000111100010000110010010000111110010000111011010000110000010000110111010000111111010001000010010001000010010000111110010000111010010000111110010000111011010000111000010001000111010000110101010001000001010001000010010000110010010000111110010001000001010000111011010000111110010000110010010000111101010000110101010000111000010000110111010000111100010000110101010000111101010000111101010000111110010000110010010000111110010000110111010001000000010000110000010001000001010001000010010000110000010000110101010001000010010001000010010001000111010000111010010000110010010000111010010000111110010000111111010000111000010001000000010000110000010000111001010001000010010000110101010001000000010001000001010000111010010000111110010000111001010000110100010000110101010001001111010001000010010000110101010000111011010001001100010000111101010000111110010001000001010001000010010000111000010000111111010001000000010000111000010000111101010001001111010001000010010000111110010001000001010001000111010000111000010001000010010000110000010001000010010001001100010001000010010001001011010001000001010001001111010001000111010000111000010001000001010000111111010001000000010000111110010000110001010000110101010000111011010000110000010000111100010000111000010000111000010000111011010000111000010000110001010000110101010000110111010001000010010001000111010000111010010001000011010001000111010000110101010001000010010000111111010001000000010000111110010000110001010000110101010000111011010000111110010000110010010001000011010000110010010000110101010000111011010000111000010001000111010000111000010000110010010000110000010000110101010001000010010000111110010000110001010001001010010000110101010000111100010001000010010000110101010000111010010001000001010001000010010000110000010000111111010001000000010000111000010000111100010000110101010001000000010000111101010000111110010000111101010000110000010001000001010001000010010000111110010000111000010000111011010000111000010000110100010000110010010000110101010001000001010001000010010000111000010001000001010000111000010000111100010000110010010000111110010000111011010000111110010000110010010000111000010000111100010000110101010000111101010000111101010000111110010001000001010001000010010000111110010000111011010001001100010000111010010000111110010001000000010000110000010000110111010000111100010001001011010001000000010000110000010000110111010000110100010000110101010000111011010001001111010000110101010000111100010001000001010000111011010000111110010000110010010000110000010001000001010000110010010000111110010000110001010000111110010000110100010000111101010001001011010000111100010000111111010001000000010000111110010001000001010001000010010001000000010000110000010000111101010001000001010001000010010000110010010000111110010000111100010001000010010001000111010000111010010001000001010001000111010000111000010001000010010000110000010001000010010001001100010000111111010001000000010000111110010000110001010000110101010000111011010001001011010000110111010000110000010000111010010000110000010000110111010001000111010000111000010000111010010000111000010000111101010000110101010000111011010001001110010000110001010001001111010001000010010000110111010000111111010001000010010001000010010000110000010000111010010000111010010000110000010000111010010001001101010001000010010000111110010000111111010001000011010001000001010001000010010000111110010000110101010000111100010000110101010001000001010001000010010000111110010001000010010001000111010000111010010000111110010000110100010000111101010000110000010000111010010000111110010000111101010000110101010000111010010000111110010001000010010000111110010001000000010001001011010000110101010001000100010000111000010001000000010000111100010001001011010000111000010000110001010000111000010001000000010000110110010000111000010000110010010000111000010000110100010001001111010001000010010001000001010000111111010001000000010000110000010000110010010000110101010000110100010000111011010000111000010000110010010001001011010000111100010001000001010001000010010000110000010000110010010000111000010001000010010001001100010001000001010001000010010000111110010000111000010000111100010000111110010001000001010001000010010001001100010000110111010000110000010001000010010001001011010001000001010001001111010001000111010001000011010001000001010000111000010000111100010000110010010000111110010000111011010000111110010000110010010001000001010000111111010001000000010000111110010000110001010000110101010000111011010000110000010000111100010000111000010000110111010000111111010001000010010001000001010001000111010000111000010001000010010000110000010001001111010000111111010000111110010001000001010000111011010000110101010000110100010000111101010000111000010000110101010000110010010000110000010000110110010000111101010001001011010000111100010001001101010000111011010000110101010000111100010000110101010000111101010001000010010000111110010000111100010000111010010000110000010001000111010000110101010001000001010001000010010000110010010000110101010000111101010000111101010000111110010000110011010000111110010000110010010000111110010001000001010000111111010001000000010000111000010001001111010001000010010000111000010001001111010001000010010001000111010000111010010001000001010000111110010000110011010000111011010000110000010001000001010000111000010001000010010000110101010001000001010001001100010000110111010000111111010001000010010001000111010000111000010001000010010000110000010001000010010001001100010001000001010000111011010000111000010001000010010000111101010001001011010000111001010001000010010000110101010000111010010001000001010001000010010000110001010000110101010000110111010000110101010000110100010000111000010000111101010000111110010000110011010000111110010000111111010001000000010000111110010000111111010001000011010001000001010000111010010000110000010000110111010000111111010001000010010000111101010000111000010000111010010001000010010000111110010000111101010000110101010000110001010001000011010000110100010000110101010001000010010001000010010001000111010000111010010000111101010000111110010000110001010000111110010000111011010001001100010001001000010000111000010000111101010001000001010001000010010000110010010001000011010000111101010001000011010000110110010000111101010000110000010001000110010000110101010000111101010000110000010000110111010000110000010001000010010001001011010001000001010001001111010001000111010001000011010000110111010000111101010000110000010000111010010000111110010000110010010000110001010000110101010000110111010000111111010001000000010000111110010000110001010000110101010000111011010000111110010000110010010001000010010001000111010000111010

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-схема:**

# Блок G: КОМБИНАЦИОННЫЕ ШИФРЫ

## МАГМА

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

**Код программы:**

from base import alphabet, input\_for\_cipher\_short, input\_for\_cipher\_long, output\_from\_decrypted

pi0 = [12, 4, 6, 2, 10, 5, 11, 9, 14, 8, 13, 7, 0, 3, 15, 1]

pi1 = [6, 8, 2, 3, 9, 10, 5, 12, 1, 14, 4, 7, 11, 13, 0, 15]

pi2 = [11, 3, 5, 8, 2, 15, 10, 13, 14, 1, 7, 4, 12, 9, 6, 0]

pi3 = [12, 8, 2, 1, 13, 4, 15, 6, 7, 0, 10, 5, 3, 14, 9, 11]

pi4 = [7, 15, 5, 10, 8, 1, 6, 13, 0, 9, 3, 14, 11, 4, 2, 12]

pi5 = [5, 13, 15, 6, 9, 2, 12, 10, 11, 7, 8, 1, 4, 3, 14, 0]

pi6 = [8, 14, 2, 5, 6, 9, 1, 12, 15, 4, 11, 0, 13, 10, 3, 7]

pi7 = [1, 7, 14, 13, 0, 5, 8, 3, 4, 15, 10, 6, 9, 12, 11, 2]

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

MASK32 = 2 \*\* 32 – 1

def t(x):

    y = 0

    for i in reversed(range(8)):

        j = (x >> 4 \* i) & 0xf

        y <<= 4

        y ^= pi[i][j]

    return y

def rot11(x):

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

def g(x, k):

    return rot11(t((x + k) % 2 \*\* 32))

def split(x):

    L = x >> 32

    R = x & MASK32

    return (L, R)

def join(L, R):

    return (L << 32) ^ R

def magma\_key\_schedule(k):

    keys = []

    for i in reversed(range(8)):

        keys.append((k >> (32 \* i)) & MASK32)

    for i in range(8):

        keys.append(keys[i])

    for i in range(8):

        keys.append(keys[i])

    for i in reversed(range(8)):

        keys.append(keys[i])

    return keys

def magma\_encrypt(x, k):

    keys = magma\_key\_schedule(k)

    (L, R) = split(x)

    for i in range(31):

        (L, R) = (R, L ^ g(R, keys[i]))

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

def magma\_decrypt(x, k):

    keys = magma\_key\_schedule(k)

    keys.reverse()

    (L, R) = split(x)

    for i in range(31):

        (L, R) = (R, L ^ g(R, keys[i]))

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

key = int('ffeeddccbbaa99887766554433221100f0f1f2f3f4f5f6f7f8f9fafbfcfdfeff', 16)

i = 0

text\_short = input\_for\_cipher\_short()

encr\_short = []

while (i < len(text\_short)):

    text = text\_short[i:i+4].encode().hex()

    text = int(text, 16)

    text = text % 2\*\*64

    pt = text

    ct = magma\_encrypt(pt, key)

    encr\_short.append(ct)

    i += 4

decr\_short = []

for i in encr\_short:

    dt = magma\_decrypt(i, key)

    decr\_short.append(bytes.fromhex(hex(dt)[2::]).decode('utf-8'))

i = 0

text\_long = input\_for\_cipher\_long()

encr\_long = []

while (i < len(text\_long)):

    text = text\_long[i:i+4].encode().hex()

    text = int(text, 16)

    text = text % 2\*\*64

    pt = text

    ct = magma\_encrypt(pt, key)

    encr\_long.append(ct)

    i += 4

decr\_long = []

for i in encr\_long:

    dt = magma\_decrypt(i, key)

    decr\_long.append(bytes.fromhex(hex(dt)[2::]).decode('utf-8'))

print(f'''

МАГМА:

КЛЮЧ:

{key}

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

{encr\_short}

Расшифрованный текст:

{output\_from\_decrypted(''.join(decr\_short))}

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

{encr\_long}

Расшифрованный текст:

{output\_from\_decrypted(''.join(decr\_long))}

''')

**Тестирование:**

/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab07\_17\_magma.py

МАГМА:

КЛЮЧ:

115761816795685524522806652725025505786200410505847444308688553892001406123775

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

[18432907413224455314, 10996816857283808610, 1603220777717569738, 6798339374272425273, 625275379878570582, 12897841916972840738, 12693135464871535956, 6338906346771095526, 2952080121925535959, 10585345143535530769]

Расшифрованный текст:

время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

[8343875602038808058, 8041676007725686027, 7453366501928099122, 13622418350652450349, 13257185717927971463, 3967957328028735447, 6303045106133971510, 17694607534926283087, 3147373830694159915, 9390122889656481141, 15485378634152172683, 11157654134498461325, 9103621146938759596, 18167786785284461467, 4720588269896140616, 13198230271257633374, 290271297514756748, 6383750138269828222, 6966411025704842352, 1184081237020962173, 5518298843322725716, 1121075660661397656, 10048741617664022091, 18363685243888377546, 95605295917991907, 17057631210449679753, 16551488750349984268, 2025585132759738150, 12949061887509852732, 9329546944062831470, 3418244764798143790, 15295462055836379806, 5155680359283034501, 9451937380501232912, 7328220556957620599, 3614937074236676557, 4589181499513045879, 13803166415433109120, 18278897981036660435, 4019936986318215394, 12394731635219602309, 5416248858033923816, 463232828950025933, 15989759834534178049, 13360939273674915028, 8528608555291906000, 10293453731533650333, 13238975415012678998, 1261680965272169368, 10483829524585249195, 15703560336104773581, 2928612788789974786, 12370719347969566168, 15498018845651124401, 7975202005615435922, 1001640210394968931, 9482390876876309460, 10306886503249707528, 3432371120083104576, 13284723522159984134, 4732957936308484548, 18145150074698797421, 7665912658016440955, 17186319045640305491, 13257185717927971463, 3967957328028735447, 6303045106133971510, 1163720297091009342, 2346699712756694809, 10356924971308547229, 13990735642799502546, 16656837303721777190, 3620970884711728977, 9443487292630364151, 7691049094682026909, 916112073177174944, 15797466941996902229, 16777509928585489639, 6303121470213798711, 11674884899395813190, 18046574429007017128, 13077766683806041775, 6569214857954439428, 13257185717927971463, 3967957328028735447, 6303045106133971510, 18131068278186858513, 16002418093945299027, 2908114868778074520, 11794392602089101467, 1199691821864277191, 3076938350476950006, 13450817471486594379, 10728741519359495825, 9269189682480091597, 14111778268298096419, 2057896382844501704, 10293453731533650333, 17204212421686848739, 5366206226868800702, 8248466997448536722, 10088810565957275850, 2254355090243742257, 8371896884399298133, 10985518332543285851, 17941711041904849701, 7740200207943279354, 10750419551538638209, 2629297776519959089, 2225742027093608502, 287244974227193751, 17712479779128577940, 9222296997056994615, 973788500067181560, 3393954245381334839, 10641483480207737426, 16269104980183698483, 16250235473790191753, 2700688729269881630, 8411607884348754050, 7636448048795956798, 692746442652315515, 11343070527907873916, 629521920599350430, 14623128418242633989, 5705247633908105959, 17910568456840661856, 16201319981229519019, 13769298882463179721, 7421154077805485456, 9312183877157555560, 7691049094682026909, 916112073177174944, 15135398776342705892, 15289260373132081671, 7126379060141568292, 1725036945967272673, 17688755856354297123, 138113508643393643, 14127475102588350875, 4151016356341159175, 9841467487393741636, 3187959327495924697, 15728081324422447459, 16055605958376054530, 3572266014017346359, 3223827000587680474, 8253733500166311251, 9140508542287809951, 1790032636239692933, 14268543881935307321, 7342482303792008383, 17677570963908781400, 13355264069952555938, 88597409000583480, 17257030184527390406, 12715322944275189718, 4311305660788670662, 7008462056628757749, 5705247633908105959, 780471545777017295, 7181337429584918154, 17602858510462705198, 14738061815484779755, 540299310016161264, 10182683375894303875, 8026343204551337982, 17660378940581534682, 13825198160625090606, 4720588269896140616, 5730527409401222056, 17941293450735671408, 4342366351984147086, 5313276448158165379, 1745024168316228346, 2528136519964988665, 3547755811857389063, 10306886503249707528, 4526494865159228242, 7342482303792008383, 2908114868778074520, 7471680038596777270, 16412799349039310306, 17456146687817481297, 2894321475946271648, 6923158161737971479, 10027397076122517884, 11785376343832896276, 4548889844418068535, 9068186136604959843, 17006372751687582512, 9582236842043709584, 2756346197016409968, 12319862187853716832, 14738061815484779755, 3597986015127692257, 6006230909502237940, 8049089039016717789, 6157047194034613233, 6958374052664244197, 16431478885293681293, 8935011217054641103, 4493811008412807767, 2950821783567146896, 1855179961465792476, 6714524409280649005, 13570818025122198173, 14348383310813638904, 4453067278648320496, 3438244984932255310, 9108046240042143694, 151270379504886441, 10183946694284410805, 580256645846205130, 5356452764985505955, 1949584005305844760, 5054509616596506897, 806561947066774048, 14232614625982377749, 4486402459089481420, 6142067374720541917, 11774712973292423210, 10305629430321463988, 3547755811857389063, 10306886503249707528, 15489555906471766183, 15441006425873291865, 2659805083408552087, 12582666581260422456, 541882649396588128, 4886609086442669621, 5561196856767141965, 7731325888395147745, 1096626906536588642, 763947604362793674, 12798794780496052712, 8299242569789435168, 6766061707386831762, 8971528550958227067, 11254181301937016096, 4106793478691170180, 5812305374465111022, 14122431208396972645, 8756880839685289505, 1840145209499476946, 17204212421686848739, 12702832949266190125, 12312436585117905032, 2731161190011158263, 4720588269896140616, 13119149454917978330, 9822108851446653348, 4085677428333094007, 12601379616160262740, 10265087333462641478, 8770956180316534333, 2746212430076756819, 10780803050552638165, 1906124277192103447, 8318790287988703211, 9451937380501232912, 7154010589201716226, 6593238725915939856, 16381960257093506421, 16805496448556233763, 13257185717927971463, 3534238871897283987, 18436104050552264826, 3237166291520387028, 14738061815484779755, 540299310016161264, 10585345143535530769]

Расшифрованный текст:

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

**Блок-схема:**

# БЛОК H: АСИММЕТРИЧНЫЕ ШИФРЫ



## RSA

Берутся два очень больших простых числа P и Q и находятся произведение простых чисел N=P×Q и функция Эйлера от этого произведения φ(N)=(P-1)×(Q-1). Выбирается случайное целое число E, взаимно простое с φ(N), и вычисляется D=(1 MOD φ(N))/E. E и N публикуются как открытый ключ, D сохраняется в тайне. Шифрование: Если M — сообщение, то шифртекст Ci получается последовательным шифрованием каждой шифрвеличины Mi возведением ее в степень E по модулю N: Ci = Mi EMOD N. Расшифрование: Получатель расшифровывает сообщение, возводя последовательно Сi в степень D по модулю N: Mi = Ci D MOD N.

**Код программы:**

# -\*- coding:utf-8 -\*-

from base import alphabet, input\_for\_cipher\_short, input\_for\_cipher\_long, output\_from\_decrypted

import random

def gcd(a, b):

    while b != 0:

        a, b = b, a % b

    return a

def multiplicative\_inverse(e, r):

    for i in range(r):

        if((e\*i) % r == 1):

            return i

def is\_prime(num):

    if num == 2:

        return True

    if num < 2 or num % 2 == 0:

        return False

    for n in range(3, int(num\*\*0.5)+2, 2):

        if num % n == 0:

            return False

    return True

def generate\_keypair(p, q):

    if not (is\_prime(p) and is\_prime(q)):

        raise ValueError('Оба числа должны быть простыми.')

    elif p == q:

        raise ValueError('p и q не могут быть равны друг другу')

    n = p \* q

    phi = (p-1) \* (q-1)

    e = random.randrange(1, phi)

    g = gcd(e, phi)

    while g != 1:

        e = random.randrange(1, phi)

        g = gcd(e, phi)

    d = multiplicative\_inverse(e, phi)

    return ((e, n), (d, n))

def encrypt(pk, plaintext):

    key, n = pk

    cipher = [(ord(char) \*\* key) % n for char in plaintext]

    return cipher

def decrypt(pk, ciphertext):

    key, n = pk

    plain = [chr((char \*\* key) % n) for char in ciphertext]

    return ''.join(plain)

p = 107

q = 109

public, private = generate\_keypair(p, q)

message\_short = input\_for\_cipher\_short()

encrypted\_short = encrypt(private, message\_short)

print\_enc\_short = ''.join([str(x) for x in encrypted\_short])

decrypted\_short = decrypt(public, encrypted\_short)

message\_long = input\_for\_cipher\_long()

encrypted\_long = encrypt(private, message\_long)

print\_enc\_long = ''.join([str(x) for x in encrypted\_long])

decrypted\_long = decrypt(public, encrypted\_long)

print(f'''

RSA:

Ключ:

p={p} q={q}

Публичный: {public}

Приватный: {private}

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

{print\_enc\_short}

Расшифрованный текст:

{output\_from\_decrypted(decrypted\_short)}

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

{print\_enc\_long}

Расшифрованный текст:

{output\_from\_decrypted(decrypted\_long)}

''')

**Тестирование:**

/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab08\_21\_rsa.py ─╯

RSA:

Ключ:

p=107 q=109

Публичный: (5441, 11663)

Приватный: (4025, 11663)

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

75927171884793859901301256591002856597171188180418875921065418863381002818041887592106548477884731961075410029100286456884718046338759288472548100181002864562548

Расшифрованный текст:

время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

75926338100285659717118893858847717121791002810018100284390188847710018100281065421799901645610029217918893857592633818046338759210028645625482656100286338107546338217910028100181002863386456847763389385100181804884784774390254818865681002888472548217910028301256591002863385659100281889385100181804439084776338565963381075432796338107549901669818865681075418049901254810018717110028633864568847254810028633875921001871716338759275921888477100288847717184778847100281881804188938510018653310018301218884771001832791881804188107541804990184778847310763381804439041311883279188847785256338717193851001894671886338847784771065432795659100293107180418825481001894671886568100286456254875921002810018254863389385100288847254821791002888477171884710754254863383107106547592100188847100283107633818048847884710754759210029327918818041881002871716002327910018310730121001894678847759218863383107106546456847763386338107541888477565963381075430121001865336338180463387592633825481002864562548847763389385633831968477633818831078847301284778847653363381002864562548847710018100281065421799901645610029217918893857592633818046338759271718847254863389385884784771075463387592100188477633818821795659633818044390301263387592100181002843906338107541888477188180418810754759210018254818041151964561001818863381075484771002925481001871711002818884771002910028645625481002888472548217910028847710018100281065421799901645610029217918893857592633818046338759226561002863382179254863381804439025486338565971711889385884771718477633821791804633875921002864562548217910028100181002818821791002818825481001856596338254810018301210654759210018884710028301256591002864561002863381002810654217999016456100187592254818041151964561001888471002875922179884731079901217910028633856599901100284390107548847217999011002818818041881075475928847217910028188217918046338759221797171884710754847788476568759288471804188645618884771065410028645625488477633830125659100288847217918041883012180463381002956596338100287171884731071804990110028439056597171884710754180463386533100189385188301256591002821796338115193012100189385188188107547171100296533188938518864561001821791002899019385188717188476456188847710018633810754188847718818041881075475921001821791889385759263381804100183012565910028100286338254863381804188645688472179100287592633821791804633875928477884718830129385884784778477633875926338301271711001821791002810018884710028100286456254875922548633856591887171100186568100288847717121792548633865681075488479901100288847180443908477633821791002818856597171188847799011002863382179645618810028100181002843901002810654217999016456188217956597171633831078847180410018938518818818041883107884730121002864562548100296456884710028565971716338310788471804633875921002975928847180418864561887592100188847100286338310756278847938510028884725482179100281001856597171188938588477171847763388477100182179100286338188180418810754759288472179100281882179188938575926338180463387592188938588478477847763382179100286338180443902548633871711001830129385106547171100183012107548847180499018847938521791804633875921001821797592633831076338107548477106549385565971716338217910028717110018847721791002875926338938510028645625482179645618810028100181002843905659717163383107884718041065430121001825481001830126456188254818884778847180411519310799011002830125659100281002810018254825481001825482656100286338565910029217910028633888479385884721791002863381002864562548633810754847710018254863388477884725486338100286338717110654884785251887171938510654188310718871713196188759218810754990110028217956597171100187592884710754180418875921065493852179100281001875921881002843902179100286338188938563382179100284390301210018100281065421799901645610029217918893857592633818046338759221795659717163383107884718041001893851883012565910028217964561881002810018990156596338217918048847107548477188884775921001831968477106549385265618048847938588478477100286338938525481001864568847217910028759288478477847763386533633875926338217956597171188990110028188990110028645625482179633865331804100182179188100288847217943903012565910028645618810028100181002843902179180418810028847710654656810028884725482179100283107884730128847107541888477633865336338565971716338565910029217925481001830125659100288477188254810028633884778847310710029107548847100281002864562548847763383107633818044390413118884772179100287592100298477100293196847710018946788478477100183012100181002810654217999016456100293012847710018254863387592310788473012565971716338310788471804633875921002864562548

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-схема:**

## Elgamal

Пусть М — исходное сообщение, состоящее из последовательных шифрвеличин Mi

Шифрование: 1. Выбирается большое простое число p, p > Mi. 2. Выбираются числа x и g так, что 1 < x < p, 1 < g < p. 3. Вычисляется Таким образом, открытые ключи: p, g, y. Секретный ключ: x. 4. Выбираются случайные секретные числа ki (рандомизаторы), взаимно простые с функцией Эйлера от числа р (φ(p) = p − 1): (ki , φ(p)) = 1. 5. Вычисляется для каждой шифрвеличины Mi : ai = gki (mod p) bi = yki Mi (mod p) Пара чисел a и b является шифробозначением. Последовательность aibi образует шифртекст. Длина шифртекста получается длинней открытого текста вдвое.

Расшифрование: Для расшифрования a и b, вычисляется: Mi ≡ bi/ ai x (mod p) Т.е. решается сравнение относительно М: Для этого надо избавиться от знаменателя: умножаем левую и правую часть сравнения на ах и решаем полученное сравнение относительно M, пользуясь расширенным алгоритмом Евклида: ax M ≡ b (mod p).

**Код программы:**

import random

from math import pow

from base import alphabet, input\_for\_cipher\_short, input\_for\_cipher\_long, output\_from\_decrypted

a = random.randint(2, 10)

def gcd(a, b):

    if a < b:

        return gcd(b, a)

    elif a % b == 0:

        return b

    else:

        return gcd(b, a % b)

def gen\_key(q):

    key = random.randint(pow(10, 20), q)

    while gcd(q, key) != 1:

        key = random.randint(pow(10, 20), q)

    return key

def power(a, b, c):

    x = 1

    y = a

    while b > 0:

        if b % 2 == 0:

            x = (x\*y) % c

        y = (y\*y) % c

        b = int(b/2)

    return x % c

def encryption(msg, q, h, g):

    ct = []

    k = gen\_key(q)

    s = power(h, k, q)

    p = power(g, k, q)

    for i in range(0, len(msg)):

        ct.append(msg[i])

    for i in range(0, len(ct)):

        ct[i] = s\*ord(ct[i])

    return ct, p

def decryption(ct, p, key, q):

    pt = []

    h = power(p, key, q)

    for i in range(0, len(ct)):

        pt.append(chr(int(ct[i]/h)))

    return pt

msg\_short = input\_for\_cipher\_short()

msg\_long = input\_for\_cipher\_long()

p = random.randint(pow(10, 20), pow(10, 30))

x = random.randint(2, p)

g = gen\_key(p)

y = power(x, g, p)

ct\_sh, pp\_sh = encryption(msg\_short, p, y, x)

pt\_sh = decryption(ct\_sh, pp\_sh, g, p)

d\_msg\_sh = ''.join(pt\_sh)

ct\_ln, pp\_ln = encryption(msg\_long, p, y, x)

pt\_ln = decryption(ct\_ln, pp\_ln, g, p)

d\_msg\_ln = ''.join(pt\_ln)

print(f'''

Elgamal:

Ключ:

p={p} x={x} g={g} y={y}

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

{ct\_sh}

Расшифрованный текст:

{output\_from\_decrypted(d\_msg\_sh)}

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

{ct\_ln}

Расшифрованный текст:

{output\_from\_decrypted(d\_msg\_ln)}

''')

**Тестирование:**

/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab08\_22\_elgamal.py

Elgamal:

Ключ:

p=824845493538558886603996765133 x=807857891093028247653418807675 g=748858366366969210000719847844 y=602207897365468982242151658873

КОРОТКИЙ ТЕКСТ:

Зашифрованный текст:

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

Расшифрованный текст:

время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:

Зашифрованный текст:

[455455369431278054196736098674214, 460544256240566077148654937765546, 462240551843662084799294550795990, 460968330141340079061314841023157, 461392404042114080973974744280768, 457999812835922065672695518219880, 459696108439018073323335131250324, 456727591133600059934715808447047, 461392404042114080973974744280768, 461816477942888082886634647538379, 462240551843662084799294550795990, 454607221629730050371416292158992, 462240551843662084799294550795990, 466481290851402103925893583372100, 457999812835922065672695518219880, 460120182339792075235995034507935, 454607221629730050371416292158992, 462240551843662084799294550795990, 466057216950628102013233680114489, 461816477942888082886634647538379, 467753512553724109663873293144933, 464360921347532094362594067084045, 462664625744436086711954454053601, 461816477942888082886634647538379, 457999812835922065672695518219880, 459696108439018073323335131250324, 455455369431278054196736098674214, 460544256240566077148654937765546, 459272034538244071410675227992713, 460544256240566077148654937765546, 455455369431278054196736098674214, 462240551843662084799294550795990, 464360921347532094362594067084045, 458847960637470069498015324735102, 466905364752176105838553486629711, 462240551843662084799294550795990, 460544256240566077148654937765546, 456303517232826058022055905189436, 460544256240566077148654937765546, 461816477942888082886634647538379, 462240551843662084799294550795990, 454607221629730050371416292158992, 462240551843662084799294550795990, 460544256240566077148654937765546, 464360921347532094362594067084045, 460120182339792075235995034507935, 460544256240566077148654937765546, 459696108439018073323335131250324, 454607221629730050371416292158992, 459272034538244071410675227992713, 456727591133600059934715808447047, 460120182339792075235995034507935, 466481290851402103925893583372100, 458847960637470069498015324735102, 457999812835922065672695518219880, 458423886736696067585355421477491, 462240551843662084799294550795990, 456727591133600059934715808447047, 458847960637470069498015324735102, 461816477942888082886634647538379, 462240551843662084799294550795990, 457575738935148063760035614962269, 460968330141340079061314841023157, 462240551843662084799294550795990, 460544256240566077148654937765546, 460968330141340079061314841023157, 462240551843662084799294550795990, 457999812835922065672695518219880, 459696108439018073323335131250324, 454607221629730050371416292158992, 459272034538244071410675227992713, 466481290851402103925893583372100, 460120182339792075235995034507935, 460544256240566077148654937765546, 460968330141340079061314841023157, 460544256240566077148654937765546, 456303517232826058022055905189436, 463512773545984090537274260568823, 460544256240566077148654937765546, 456303517232826058022055905189436, 467753512553724109663873293144933, 465209069149080098187913873599267, 457999812835922065672695518219880, 458423886736696067585355421477491, 456303517232826058022055905189436, 459272034538244071410675227992713, 467753512553724109663873293144933, 458847960637470069498015324735102, 454607221629730050371416292158992, 461392404042114080973974744280768, 462240551843662084799294550795990, 460544256240566077148654937765546, 464360921347532094362594067084045, 456727591133600059934715808447047, 458847960637470069498015324735102, 462240551843662084799294550795990, 460544256240566077148654937765546, 455455369431278054196736098674214, 454607221629730050371416292158992, 461392404042114080973974744280768, 460544256240566077148654937765546, 455455369431278054196736098674214, 455455369431278054196736098674214, 457999812835922065672695518219880, 460120182339792075235995034507935, 462240551843662084799294550795990, 456727591133600059934715808447047, 461392404042114080973974744280768, 460120182339792075235995034507935, 456727591133600059934715808447047, 462240551843662084799294550795990, 457999812835922065672695518219880, 459272034538244071410675227992713, 457999812835922065672695518219880, 459696108439018073323335131250324, 454607221629730050371416292158992, 455879443332052056109396001931825, 454607221629730050371416292158992, 457575738935148063760035614962269, 457999812835922065672695518219880, 460120182339792075235995034507935, 454607221629730050371416292158992, 463512773545984090537274260568823, 457999812835922065672695518219880, 459272034538244071410675227992713, 457999812835922065672695518219880, 456303517232826058022055905189436, 459272034538244071410675227992713, 467753512553724109663873293144933, 460120182339792075235995034507935, 456727591133600059934715808447047, 455031295530504052284076195416603, 460544256240566077148654937765546, 459272034538244071410675227992713, 466481290851402103925893583372100, 464784995248306096275253970341656, 457999812835922065672695518219880, 463512773545984090537274260568823, 457999812835922065672695518219880, 460120182339792075235995034507935, 463088699645210088624614357311212, 460544256240566077148654937765546, 461392404042114080973974744280768, 459696108439018073323335131250324, 454607221629730050371416292158992, 463936847446758092449934163826434, 457999812835922065672695518219880, 460544256240566077148654937765546, 460120182339792075235995034507935, 460120182339792075235995034507935, 466057216950628102013233680114489, 463512773545984090537274260568823, 460968330141340079061314841023157, 462664625744436086711954454053601, 455031295530504052284076195416603, 459272034538244071410675227992713, 457999812835922065672695518219880, 458847960637470069498015324735102, 454607221629730050371416292158992, 463936847446758092449934163826434, 457999812835922065672695518219880, 458423886736696067585355421477491, 462240551843662084799294550795990, 464360921347532094362594067084045, 458847960637470069498015324735102, 455455369431278054196736098674214, 462240551843662084799294550795990, 454607221629730050371416292158992, 458847960637470069498015324735102, 460544256240566077148654937765546, 459696108439018073323335131250324, 462240551843662084799294550795990, 456727591133600059934715808447047, 458847960637470069498015324735102, 461816477942888082886634647538379, 462240551843662084799294550795990, 456727591133600059934715808447047, 461392404042114080973974744280768, 456727591133600059934715808447047, 456303517232826058022055905189436, 458847960637470069498015324735102, 460544256240566077148654937765546, 455031295530504052284076195416603, 466057216950628102013233680114489, 455455369431278054196736098674214, 454607221629730050371416292158992, 456727591133600059934715808447047, 462240551843662084799294550795990, 455031295530504052284076195416603, 460544256240566077148654937765546, 459272034538244071410675227992713, 456727591133600059934715808447047, 456727591133600059934715808447047, 456303517232826058022055905189436, 455455369431278054196736098674214, 462664625744436086711954454053601, 463512773545984090537274260568823, 457999812835922065672695518219880, 459272034538244071410675227992713, 457999812835922065672695518219880, 462240551843662084799294550795990, 461392404042114080973974744280768, 468601660355272113489193099660155, 463512773545984090537274260568823, 454607221629730050371416292158992, 455031295530504052284076195416603, 457575738935148063760035614962269, 454607221629730050371416292158992, 463936847446758092449934163826434, 456727591133600059934715808447047, 455455369431278054196736098674214, 457999812835922065672695518219880, 460544256240566077148654937765546, 455031295530504052284076195416603, 466057216950628102013233680114489, 464360921347532094362594067084045, 460120182339792075235995034507935, 460544256240566077148654937765546, 460544256240566077148654937765546, 456303517232826058022055905189436, 457999812835922065672695518219880, 460120182339792075235995034507935, 460968330141340079061314841023157, 460544256240566077148654937765546, 456303517232826058022055905189436, 457575738935148063760035614962269, 454607221629730050371416292158992, 455879443332052056109396001931825, 460544256240566077148654937765546, 459272034538244071410675227992713, 460544256240566077148654937765546, 455455369431278054196736098674214, 460544256240566077148654937765546, 458847960637470069498015324735102, 462240551843662084799294550795990, 464360921347532094362594067084045, 458847960637470069498015324735102, 460120182339792075235995034507935, 460544256240566077148654937765546, 459696108439018073323335131250324, 460544256240566077148654937765546, 457151665034374061847375711704658, 460120182339792075235995034507935, 460544256240566077148654937765546, 457999812835922065672695518219880, 455031295530504052284076195416603, 456727591133600059934715808447047, 457575738935148063760035614962269, 460120182339792075235995034507935, 456727591133600059934715808447047, 455879443332052056109396001931825, 460544256240566077148654937765546, 462240551843662084799294550795990, 464360921347532094362594067084045, 458847960637470069498015324735102, 460120182339792075235995034507935, 454607221629730050371416292158992, 462240551843662084799294550795990, 466057216950628102013233680114489, 461816477942888082886634647538379, 467753512553724109663873293144933, 464360921347532094362594067084045, 462664625744436086711954454053601, 461816477942888082886634647538379, 457999812835922065672695518219880, 459696108439018073323335131250324, 455455369431278054196736098674214, 460544256240566077148654937765546, 459272034538244071410675227992713, 460544256240566077148654937765546, 455455369431278054196736098674214, 461392404042114080973974744280768, 456727591133600059934715808447047, 458847960637470069498015324735102, 460544256240566077148654937765546, 459696108439018073323335131250324, 456727591133600059934715808447047, 460120182339792075235995034507935, 456303517232826058022055905189436, 460544256240566077148654937765546, 455455369431278054196736098674214, 454607221629730050371416292158992, 460120182339792075235995034507935, 460544256240566077148654937765546, 457999812835922065672695518219880, 461816477942888082886634647538379, 460968330141340079061314841023157, 460544256240566077148654937765546, 459272034538244071410675227992713, 466481290851402103925893583372100, 457575738935148063760035614962269, 460544256240566077148654937765546, 455455369431278054196736098674214, 454607221629730050371416292158992, 462240551843662084799294550795990, 466481290851402103925893583372100, 460544256240566077148654937765546, 456303517232826058022055905189436, 457999812835922065672695518219880, 460120182339792075235995034507935, 457999812835922065672695518219880, 459272034538244071410675227992713, 457999812835922065672695518219880, 456303517232826058022055905189436, 455455369431278054196736098674214, 454607221629730050371416292158992, 458847960637470069498015324735102, 459272034538244071410675227992713, 467329438652950107751213389887322, 464360921347532094362594067084045, 454607221629730050371416292158992, 457999812835922065672695518219880, 460544256240566077148654937765546, 456303517232826058022055905189436, 460120182339792075235995034507935, 462664625744436086711954454053601, 458847960637470069498015324735102, 454607221629730050371416292158992, 461392404042114080973974744280768, 462240551843662084799294550795990, 457999812835922065672695518219880, 460120182339792075235995034507935, 462664625744436086711954454053601, 462240551843662084799294550795990, 464360921347532094362594067084045, 458847960637470069498015324735102, 462240551843662084799294550795990, 456727591133600059934715808447047, 458847960637470069498015324735102, 461816477942888082886634647538379, 462240551843662084799294550795990, 460120182339792075235995034507935, 454607221629730050371416292158992, 462240551843662084799294550795990, 466057216950628102013233680114489, 461816477942888082886634647538379, 467753512553724109663873293144933, 464360921347532094362594067084045, 462664625744436086711954454053601, 461816477942888082886634647538379, 457999812835922065672695518219880, 459696108439018073323335131250324, 455455369431278054196736098674214, 460544256240566077148654937765546, 459272034538244071410675227992713, 460544256240566077148654937765546, 455455369431278054196736098674214, 466905364752176105838553486629711, 462240551843662084799294550795990, 460544256240566077148654937765546, 461816477942888082886634647538379, 458847960637470069498015324735102, 460544256240566077148654937765546, 459272034538244071410675227992713, 466481290851402103925893583372100, 458847960637470069498015324735102, 460544256240566077148654937765546, 460968330141340079061314841023157, 461392404042114080973974744280768, 457999812835922065672695518219880, 459696108439018073323335131250324, 456727591133600059934715808447047, 461392404042114080973974744280768, 460120182339792075235995034507935, 460544256240566077148654937765546, 461816477942888082886634647538379, 459272034538244071410675227992713, 460544256240566077148654937765546, 455455369431278054196736098674214, 462240551843662084799294550795990, 464360921347532094362594067084045, 458847960637470069498015324735102, 461816477942888082886634647538379, 462240551843662084799294550795990, 454607221629730050371416292158992, 462240551843662084799294550795990, 457999812835922065672695518219880, 461816477942888082886634647538379, 462240551843662084799294550795990, 457999812835922065672695518219880, 458847960637470069498015324735102, 454607221629730050371416292158992, 460968330141340079061314841023157, 460544256240566077148654937765546, 458847960637470069498015324735102, 454607221629730050371416292158992, 457575738935148063760035614962269, 466057216950628102013233680114489, 455455369431278054196736098674214, 454607221629730050371416292158992, 456727591133600059934715808447047, 462240551843662084799294550795990, 457575738935148063760035614962269, 460968330141340079061314841023157, 462240551843662084799294550795990, 464360921347532094362594067084045, 462240551843662084799294550795990, 460544256240566077148654937765546, 462240551843662084799294550795990, 466057216950628102013233680114489, 461816477942888082886634647538379, 467753512553724109663873293144933, 464360921347532094362594067084045, 454607221629730050371416292158992, 455455369431278054196736098674214, 458847960637470069498015324735102, 459272034538244071410675227992713, 467329438652950107751213389887322, 464360921347532094362594067084045, 454607221629730050371416292158992, 456727591133600059934715808447047, 462240551843662084799294550795990, 455455369431278054196736098674214, 461816477942888082886634647538379, 456727591133600059934715808447047, 455031295530504052284076195416603, 467753512553724109663873293144933, 461816477942888082886634647538379, 462240551843662084799294550795990, 460544256240566077148654937765546, 460968330141340079061314841023157, 467753512553724109663873293144933, 462240551843662084799294550795990, 466481290851402103925893583372100, 456303517232826058022055905189436, 456727591133600059934715808447047, 461816477942888082886634647538379, 467753512553724109663873293144933, 462240551843662084799294550795990, 457999812835922065672695518219880, 459272034538244071410675227992713, 457999812835922065672695518219880, 456303517232826058022055905189436, 455455369431278054196736098674214, 456727591133600059934715808447047, 461816477942888082886634647538379, 462240551843662084799294550795990, 457999812835922065672695518219880, 461816477942888082886634647538379, 459272034538244071410675227992713, 460544256240566077148654937765546, 455455369431278054196736098674214, 461816477942888082886634647538379, 461392404042114080973974744280768, 456727591133600059934715808447047, 456303517232826058022055905189436, 460120182339792075235995034507935, 456727591133600059934715808447047, 458423886736696067585355421477491, 455455369431278054196736098674214, 456727591133600059934715808447047, 459272034538244071410675227992713, 457999812835922065672695518219880, 464360921347532094362594067084045, 457999812835922065672695518219880, 460120182339792075235995034507935, 466057216950628102013233680114489, 462240551843662084799294550795990, 464360921347532094362594067084045, 458847960637470069498015324735102, 460120182339792075235995034507935, 460544256240566077148654937765546, 457575738935148063760035614962269, 460968330141340079061314841023157, 462240551843662084799294550795990, 456727591133600059934715808447047, 461816477942888082886634647538379, 459272034538244071410675227992713, 457999812835922065672695518219880, 457575738935148063760035614962269, 459272034538244071410675227992713, 460544256240566077148654937765546, 462664625744436086711954454053601, 460968330141340079061314841023157, 460544256240566077148654937765546, 462240551843662084799294550795990, 461392404042114080973974744280768, 456727591133600059934715808447047, 455031295530504052284076195416603, 459272034538244071410675227992713, 467753512553724109663873293144933, 462240551843662084799294550795990, 466481290851402103925893583372100, 460968330141340079061314841023157, 461392404042114080973974744280768, 456727591133600059934715808447047, 456303517232826058022055905189436, 459272034538244071410675227992713, 460544256240566077148654937765546, 455879443332052056109396001931825, 454607221629730050371416292158992, 459696108439018073323335131250324, 457999812835922065672695518219880, 457575738935148063760035614962269, 460968330141340079061314841023157, 462240551843662084799294550795990, 461816477942888082886634647538379, 460544256240566077148654937765546, 467329438652950107751213389887322, 457575738935148063760035614962269, 454607221629730050371416292158992, 459696108439018073323335131250324, 457999812835922065672695518219880, 457999812835922065672695518219880, 456303517232826058022055905189436, 461392404042114080973974744280768, 462664625744436086711954454053601, 455879443332052056109396001931825, 457999812835922065672695518219880, 459696108439018073323335131250324, 457999812835922065672695518219880, 464360921347532094362594067084045, 454607221629730050371416292158992, 461816477942888082886634647538379, 462240551843662084799294550795990, 467753512553724109663873293144933, 459696108439018073323335131250324, 457999812835922065672695518219880, 461392404042114080973974744280768, 456727591133600059934715808447047, 464360921347532094362594067084045, 457999812835922065672695518219880, 460120182339792075235995034507935, 454607221629730050371416292158992, 460544256240566077148654937765546, 456303517232826058022055905189436, 457999812835922065672695518219880, 460120182339792075235995034507935, 457999812835922065672695518219880, 459272034538244071410675227992713, 457999812835922065672695518219880, 456303517232826058022055905189436, 455455369431278054196736098674214, 454607221629730050371416292158992, 461816477942888082886634647538379, 457999812835922065672695518219880, 459696108439018073323335131250324, 455455369431278054196736098674214, 460544256240566077148654937765546, 459272034538244071410675227992713, 454607221629730050371416292158992, 457575738935148063760035614962269, 460968330141340079061314841023157, 462240551843662084799294550795990, 462240551843662084799294550795990, 460544256240566077148654937765546, 458847960637470069498015324735102, 460544256240566077148654937765546, 459272034538244071410675227992713, 457999812835922065672695518219880, 464360921347532094362594067084045, 456727591133600059934715808447047, 461816477942888082886634647538379, 462240551843662084799294550795990, 455455369431278054196736098674214, 460544256240566077148654937765546, 461816477942888082886634647538379, 459272034538244071410675227992713, 460544256240566077148654937765546, 455455369431278054196736098674214, 460120182339792075235995034507935, 456727591133600059934715808447047, 457999812835922065672695518219880, 457575738935148063760035614962269, 459696108439018073323335131250324, 456727591133600059934715808447047, 460120182339792075235995034507935, 460120182339792075235995034507935, 460544256240566077148654937765546, 455455369431278054196736098674214, 460544256240566077148654937765546, 457575738935148063760035614962269, 461392404042114080973974744280768, 454607221629730050371416292158992, 461816477942888082886634647538379, 462240551843662084799294550795990, 454607221629730050371416292158992, 456727591133600059934715808447047, 462240551843662084799294550795990, 462240551843662084799294550795990, 464360921347532094362594067084045, 458847960637470069498015324735102, 455455369431278054196736098674214, 458847960637470069498015324735102, 460544256240566077148654937765546, 460968330141340079061314841023157, 457999812835922065672695518219880, 461392404042114080973974744280768, 454607221629730050371416292158992, 458423886736696067585355421477491, 462240551843662084799294550795990, 456727591133600059934715808447047, 461392404042114080973974744280768, 461816477942888082886634647538379, 458847960637470069498015324735102, 460544256240566077148654937765546, 458423886736696067585355421477491, 456303517232826058022055905189436, 456727591133600059934715808447047, 467753512553724109663873293144933, 462240551843662084799294550795990, 456727591133600059934715808447047, 459272034538244071410675227992713, 466481290851402103925893583372100, 460120182339792075235995034507935, 460544256240566077148654937765546, 461816477942888082886634647538379, 462240551843662084799294550795990, 457999812835922065672695518219880, 460968330141340079061314841023157, 461392404042114080973974744280768, 457999812835922065672695518219880, 460120182339792075235995034507935, 467753512553724109663873293144933, 462240551843662084799294550795990, 460544256240566077148654937765546, 461816477942888082886634647538379, 464360921347532094362594067084045, 457999812835922065672695518219880, 462240551843662084799294550795990, 454607221629730050371416292158992, 462240551843662084799294550795990, 466481290851402103925893583372100, 462240551843662084799294550795990, 466057216950628102013233680114489, 461816477942888082886634647538379, 467753512553724109663873293144933, 464360921347532094362594067084045, 457999812835922065672695518219880, 461816477942888082886634647538379, 460968330141340079061314841023157, 461392404042114080973974744280768, 460544256240566077148654937765546, 455031295530504052284076195416603, 456727591133600059934715808447047, 459272034538244071410675227992713, 454607221629730050371416292158992, 459696108439018073323335131250324, 457999812835922065672695518219880, 457999812835922065672695518219880, 459272034538244071410675227992713, 457999812835922065672695518219880, 455031295530504052284076195416603, 456727591133600059934715808447047, 457575738935148063760035614962269, 462240551843662084799294550795990, 464360921347532094362594067084045, 458847960637470069498015324735102, 462664625744436086711954454053601, 464360921347532094362594067084045, 456727591133600059934715808447047, 462240551843662084799294550795990, 460968330141340079061314841023157, 461392404042114080973974744280768, 460544256240566077148654937765546, 455031295530504052284076195416603, 456727591133600059934715808447047, 459272034538244071410675227992713, 460544256240566077148654937765546, 455455369431278054196736098674214, 462664625744436086711954454053601, 455455369431278054196736098674214, 456727591133600059934715808447047, 459272034538244071410675227992713, 457999812835922065672695518219880, 464360921347532094362594067084045, 457999812835922065672695518219880, 455455369431278054196736098674214, 454607221629730050371416292158992, 456727591133600059934715808447047, 462240551843662084799294550795990, 460544256240566077148654937765546, 455031295530504052284076195416603, 465633143049854100100573776856878, 456727591133600059934715808447047, 459696108439018073323335131250324, 462240551843662084799294550795990, 456727591133600059934715808447047, 458847960637470069498015324735102, 461816477942888082886634647538379, 462240551843662084799294550795990, 454607221629730050371416292158992, 460968330141340079061314841023157, 461392404042114080973974744280768, 457999812835922065672695518219880, 459696108439018073323335131250324, 456727591133600059934715808447047, 461392404042114080973974744280768, 460120182339792075235995034507935, 460544256240566077148654937765546, 460120182339792075235995034507935, 454607221629730050371416292158992, 461816477942888082886634647538379, 462240551843662084799294550795990, 460544256240566077148654937765546, 457999812835922065672695518219880, 459272034538244071410675227992713, 457999812835922065672695518219880, 456303517232826058022055905189436, 455455369431278054196736098674214, 456727591133600059934715808447047, 461816477942888082886634647538379, 462240551843662084799294550795990, 457999812835922065672695518219880, 461816477942888082886634647538379, 457999812835922065672695518219880, 459696108439018073323335131250324, 455455369431278054196736098674214, 460544256240566077148654937765546, 459272034538244071410675227992713, 460544256240566077148654937765546, 455455369431278054196736098674214, 457999812835922065672695518219880, 459696108439018073323335131250324, 456727591133600059934715808447047, 460120182339792075235995034507935, 460120182339792075235995034507935, 460544256240566077148654937765546, 461816477942888082886634647538379, 462240551843662084799294550795990, 460544256240566077148654937765546, 459272034538244071410675227992713, 466481290851402103925893583372100, 458847960637470069498015324735102, 460544256240566077148654937765546, 461392404042114080973974744280768, 454607221629730050371416292158992, 457575738935148063760035614962269, 459696108439018073323335131250324, 466057216950628102013233680114489, 461392404042114080973974744280768, 454607221629730050371416292158992, 457575738935148063760035614962269, 456303517232826058022055905189436, 456727591133600059934715808447047, 459272034538244071410675227992713, 467753512553724109663873293144933, 456727591133600059934715808447047, 459696108439018073323335131250324, 461816477942888082886634647538379, 459272034538244071410675227992713, 460544256240566077148654937765546, 455455369431278054196736098674214, 454607221629730050371416292158992, 461816477942888082886634647538379, 455455369431278054196736098674214, 460544256240566077148654937765546, 455031295530504052284076195416603, 460544256240566077148654937765546, 456303517232826058022055905189436, 460120182339792075235995034507935, 466057216950628102013233680114489, 459696108439018073323335131250324, 460968330141340079061314841023157, 461392404042114080973974744280768, 460544256240566077148654937765546, 461816477942888082886634647538379, 462240551843662084799294550795990, 461392404042114080973974744280768, 454607221629730050371416292158992, 460120182339792075235995034507935, 461816477942888082886634647538379, 462240551843662084799294550795990, 455455369431278054196736098674214, 460544256240566077148654937765546, 459696108439018073323335131250324, 462240551843662084799294550795990, 464360921347532094362594067084045, 458847960637470069498015324735102, 461816477942888082886634647538379, 464360921347532094362594067084045, 457999812835922065672695518219880, 462240551843662084799294550795990, 454607221629730050371416292158992, 462240551843662084799294550795990, 466481290851402103925893583372100, 460968330141340079061314841023157, 461392404042114080973974744280768, 460544256240566077148654937765546, 455031295530504052284076195416603, 456727591133600059934715808447047, 459272034538244071410675227992713, 466057216950628102013233680114489, 457575738935148063760035614962269, 454607221629730050371416292158992, 458847960637470069498015324735102, 454607221629730050371416292158992, 457575738935148063760035614962269, 464360921347532094362594067084045, 457999812835922065672695518219880, 458847960637470069498015324735102, 457999812835922065672695518219880, 460120182339792075235995034507935, 456727591133600059934715808447047, 459272034538244071410675227992713, 467329438652950107751213389887322, 455031295530504052284076195416603, 467753512553724109663873293144933, 462240551843662084799294550795990, 457575738935148063760035614962269, 460968330141340079061314841023157, 462240551843662084799294550795990, 462240551843662084799294550795990, 454607221629730050371416292158992, 458847960637470069498015324735102, 458847960637470069498015324735102, 454607221629730050371416292158992, 458847960637470069498015324735102, 466905364752176105838553486629711, 462240551843662084799294550795990, 460544256240566077148654937765546, 460968330141340079061314841023157, 462664625744436086711954454053601, 461816477942888082886634647538379, 462240551843662084799294550795990, 460544256240566077148654937765546, 456727591133600059934715808447047, 459696108439018073323335131250324, 456727591133600059934715808447047, 461816477942888082886634647538379, 462240551843662084799294550795990, 460544256240566077148654937765546, 462240551843662084799294550795990, 464360921347532094362594067084045, 458847960637470069498015324735102, 460544256240566077148654937765546, 456303517232826058022055905189436, 460120182339792075235995034507935, 454607221629730050371416292158992, 458847960637470069498015324735102, 460544256240566077148654937765546, 460120182339792075235995034507935, 456727591133600059934715808447047, 458847960637470069498015324735102, 460544256240566077148654937765546, 462240551843662084799294550795990, 460544256240566077148654937765546, 461392404042114080973974744280768, 466057216950628102013233680114489, 456727591133600059934715808447047, 463088699645210088624614357311212, 457999812835922065672695518219880, 461392404042114080973974744280768, 459696108439018073323335131250324, 466057216950628102013233680114489, 457999812835922065672695518219880, 455031295530504052284076195416603, 457999812835922065672695518219880, 461392404042114080973974744280768, 457151665034374061847375711704658, 457999812835922065672695518219880, 455455369431278054196736098674214, 457999812835922065672695518219880, 456303517232826058022055905189436, 467753512553724109663873293144933, 462240551843662084799294550795990, 461816477942888082886634647538379, 460968330141340079061314841023157, 461392404042114080973974744280768, 454607221629730050371416292158992, 455455369431278054196736098674214, 456727591133600059934715808447047, 456303517232826058022055905189436, 459272034538244071410675227992713, 457999812835922065672695518219880, 455455369431278054196736098674214, 466057216950628102013233680114489, 459696108439018073323335131250324, 461816477942888082886634647538379, 462240551843662084799294550795990, 454607221629730050371416292158992, 455455369431278054196736098674214, 457999812835922065672695518219880, 462240551843662084799294550795990, 466481290851402103925893583372100, 461816477942888082886634647538379, 462240551843662084799294550795990, 460544256240566077148654937765546, 457999812835922065672695518219880, 459696108439018073323335131250324, 460544256240566077148654937765546, 461816477942888082886634647538379, 462240551843662084799294550795990, 466481290851402103925893583372100, 457575738935148063760035614962269, 454607221629730050371416292158992, 462240551843662084799294550795990, 466057216950628102013233680114489, 461816477942888082886634647538379, 467753512553724109663873293144933, 464360921347532094362594067084045, 462664625744436086711954454053601, 461816477942888082886634647538379, 457999812835922065672695518219880, 459696108439018073323335131250324, 455455369431278054196736098674214, 460544256240566077148654937765546, 459272034538244071410675227992713, 460544256240566077148654937765546, 455455369431278054196736098674214, 461816477942888082886634647538379, 460968330141340079061314841023157, 461392404042114080973974744280768, 460544256240566077148654937765546, 455031295530504052284076195416603, 456727591133600059934715808447047, 459272034538244071410675227992713, 454607221629730050371416292158992, 459696108439018073323335131250324, 457999812835922065672695518219880, 457575738935148063760035614962269, 460968330141340079061314841023157, 462240551843662084799294550795990, 461816477942888082886634647538379, 464360921347532094362594067084045, 457999812835922065672695518219880, 462240551843662084799294550795990, 454607221629730050371416292158992, 467753512553724109663873293144933, 460968330141340079061314841023157, 460544256240566077148654937765546, 461816477942888082886634647538379, 459272034538244071410675227992713, 456727591133600059934715808447047, 456303517232826058022055905189436, 460120182339792075235995034507935, 457999812835922065672695518219880, 456727591133600059934715808447047, 455455369431278054196736098674214, 454607221629730050371416292158992, 457151665034374061847375711704658, 460120182339792075235995034507935, 466057216950628102013233680114489, 459696108439018073323335131250324, 466905364752176105838553486629711, 459272034538244071410675227992713, 456727591133600059934715808447047, 459696108439018073323335131250324, 456727591133600059934715808447047, 460120182339792075235995034507935, 462240551843662084799294550795990, 460544256240566077148654937765546, 459696108439018073323335131250324, 458847960637470069498015324735102, 454607221629730050371416292158992, 464360921347532094362594067084045, 456727591133600059934715808447047, 461816477942888082886634647538379, 462240551843662084799294550795990, 455455369431278054196736098674214, 456727591133600059934715808447047, 460120182339792075235995034507935, 460120182339792075235995034507935, 460544256240566077148654937765546, 455879443332052056109396001931825, 460544256240566077148654937765546, 455455369431278054196736098674214, 460544256240566077148654937765546, 461816477942888082886634647538379, 460968330141340079061314841023157, 461392404042114080973974744280768, 457999812835922065672695518219880, 467753512553724109663873293144933, 462240551843662084799294550795990, 457999812835922065672695518219880, 467753512553724109663873293144933, 462240551843662084799294550795990, 464360921347532094362594067084045, 458847960637470069498015324735102, 461816477942888082886634647538379, 460544256240566077148654937765546, 455879443332052056109396001931825, 459272034538244071410675227992713, 454607221629730050371416292158992, 461816477942888082886634647538379, 457999812835922065672695518219880, 462240551843662084799294550795990, 456727591133600059934715808447047, 461816477942888082886634647538379, 466481290851402103925893583372100, 457575738935148063760035614962269, 460968330141340079061314841023157, 462240551843662084799294550795990, 464360921347532094362594067084045, 457999812835922065672695518219880, 462240551843662084799294550795990, 454607221629730050371416292158992, 462240551843662084799294550795990, 466481290851402103925893583372100, 461816477942888082886634647538379, 459272034538244071410675227992713, 457999812835922065672695518219880, 462240551843662084799294550795990, 460120182339792075235995034507935, 466057216950628102013233680114489, 458423886736696067585355421477491, 462240551843662084799294550795990, 456727591133600059934715808447047, 458847960637470069498015324735102, 461816477942888082886634647538379, 462240551843662084799294550795990, 455031295530504052284076195416603, 456727591133600059934715808447047, 457575738935148063760035614962269, 456727591133600059934715808447047, 456303517232826058022055905189436, 457999812835922065672695518219880, 460120182339792075235995034507935, 460544256240566077148654937765546, 455879443332052056109396001931825, 460544256240566077148654937765546, 460968330141340079061314841023157, 461392404042114080973974744280768, 460544256240566077148654937765546, 460968330141340079061314841023157, 462664625744436086711954454053601, 461816477942888082886634647538379, 458847960637470069498015324735102, 454607221629730050371416292158992, 457575738935148063760035614962269, 460968330141340079061314841023157, 462240551843662084799294550795990, 460120182339792075235995034507935, 457999812835922065672695518219880, 458847960637470069498015324735102, 462240551843662084799294550795990, 460544256240566077148654937765546, 460120182339792075235995034507935, 456727591133600059934715808447047, 455031295530504052284076195416603, 462664625744436086711954454053601, 456303517232826058022055905189436, 456727591133600059934715808447047, 462240551843662084799294550795990, 462240551843662084799294550795990, 464360921347532094362594067084045, 458847960637470069498015324735102, 460120182339792075235995034507935, 460544256240566077148654937765546, 455031295530504052284076195416603, 460544256240566077148654937765546, 459272034538244071410675227992713, 466481290851402103925893583372100, 464784995248306096275253970341656, 457999812835922065672695518219880, 460120182339792075235995034507935, 461816477942888082886634647538379, 462240551843662084799294550795990, 455455369431278054196736098674214, 462664625744436086711954454053601, 460120182339792075235995034507935, 462664625744436086711954454053601, 457151665034374061847375711704658, 460120182339792075235995034507935, 454607221629730050371416292158992, 463936847446758092449934163826434, 456727591133600059934715808447047, 460120182339792075235995034507935, 454607221629730050371416292158992, 457575738935148063760035614962269, 454607221629730050371416292158992, 462240551843662084799294550795990, 466057216950628102013233680114489, 461816477942888082886634647538379, 467753512553724109663873293144933, 464360921347532094362594067084045, 462664625744436086711954454053601, 457575738935148063760035614962269, 460120182339792075235995034507935, 454607221629730050371416292158992, 458847960637470069498015324735102, 460544256240566077148654937765546, 455455369431278054196736098674214, 455031295530504052284076195416603, 456727591133600059934715808447047, 457575738935148063760035614962269, 460968330141340079061314841023157, 461392404042114080973974744280768, 460544256240566077148654937765546, 455031295530504052284076195416603, 456727591133600059934715808447047, 459272034538244071410675227992713, 460544256240566077148654937765546, 455455369431278054196736098674214, 462240551843662084799294550795990, 464360921347532094362594067084045, 458847960637470069498015324735102]

Расшифрованный текст:

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

**Блок-схема:**

# Блок I: АЛГОРИТМЫ ЦИФРОВЫХ ПОДПИСЕЙ



## RSA

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

**Код программы:**

from math import gcd

from base import alphabet, input\_for\_cipher\_short, input\_for\_cipher\_long, output\_from\_decrypted

alphabet\_lower = {'а': 0, 'б': 1, 'в': 2, 'г': 3, 'д': 4,

                  'е': 5, 'ё': 6, 'ж': 7, 'з': 8, 'и': 9, 'й': 10,

                  'к': 11, 'л': 12, 'м': 13, 'н': 14, 'о': 15,

                  'п': 16, 'р': 17, 'с': 18, 'т': 19, 'у': 20,

                  'ф': 21, 'х': 22, 'ц': 23, 'ч': 24, 'ш': 25,

                  'щ': 26, 'ъ': 27, 'ы': 28, 'ь': 29, 'э': 30,

                  'ю': 31, 'я': 32

                  }

def IsPrime(n):

    d = 2

    while n % d != 0:

        d += 1

    return d == n

def modInverse(e, el):

    e = e % el

    for x in range(1, el):

        if ((e \* x) % el == 1):

            return x

    return 1

def check\_signature(sign\_msg, n, e):

    check = (sign\_msg\*\*e) % n

    return check

def hash\_value(n, alpha\_code\_msg):

    i = 0

    hashing\_value = 1

    while i < len(alpha\_code\_msg):

        hashing\_value = (((hashing\_value-1) + int(alpha\_code\_msg[i]))\*\*2) % n

        i += 1

    return hashing\_value

def signature\_msg(hash\_code, n, d):

    sign = (hash\_code\*\*d) % n

    return sign

def rsacipher(p, q, clearText):

    p = int(p)

    print('p: ', IsPrime(p))

    q = int(q)

    print('q: ', IsPrime(q))

    n = p \* q

    print("N =", n)

    el = (p-1) \* (q-1)

    print("El =", el)

    e = 7

    print("E =", e)

    if gcd(e, el) == 1:

        print(gcd(e, el), "E подходит")

    else:

        print(gcd(e, el), "False")

    d = modInverse(e, el)

    print("D =", d)

    print("Открытый ключ e={} n={}".format(e, n))

    print("Секретный ключ d={} n={}".format(d, n))

    msg = clearText

    msg\_list = list(msg)

    alpha\_code\_msg = list()

    for i in range(len(msg\_list)):

        alpha\_code\_msg.append(int(alphabet\_lower.get(msg\_list[i])))

    print("Длина исходного сообщения {} символов".format(len(alpha\_code\_msg)))

    hash\_code\_msg = hash\_value(n, alpha\_code\_msg)

    print("Хэш сообщения", hash\_code\_msg)

    sign\_msg = signature\_msg(hash\_code\_msg, n, d)

    print("Значение подписи: {}".format(sign\_msg))

    check\_sign = check\_signature(sign\_msg, n, e)

    print("Значение проверки хэша = {}\n".format(check\_sign))

print('ЭЦП RSA:')

print('КОРОТКИЙ ТЕКСТ:')

rsacipher('31', '7', input\_for\_cipher\_short())

print('ДЛИННЫЙ ТЕКСТ:')

rsacipher('31', '7', input\_for\_cipher\_long())

**Тестирование:**

/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab09\_24\_rsa.py

ЭЦП RSA:

КОРОТКИЙ ТЕКСТ:

p: True

q: True

N = 217

El = 180

E = 7

1 E подходит

D = 103

Открытый ключ e=7 n=217

Секретный ключ d=103 n=217

Длина исходного сообщения 39 символов

Хэш сообщения 121

Значение подписи: 100

Значение проверки хэша = 121

ДЛИННЫЙ ТЕКСТ:

p: True

q: True

N = 217

El = 180

E = 7

1 E подходит

D = 103

Открытый ключ e=7 n=217

Секретный ключ d=103 n=217

Длина исходного сообщения 1087 символов

Хэш сообщения 144

Значение подписи: 165

Значение проверки хэша = 144

**Блок-схема:**

## El Gamal

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

**Код программы:**

from math import gcd

import random

from base import alphabet, input\_for\_cipher\_short, input\_for\_cipher\_long, output\_from\_decrypted

alphavit = {'а': 0, 'б': 1, 'в': 2, 'г': 3, 'д': 4,

            'е': 5, 'ё': 6, 'ж': 7, 'з': 8, 'и': 9, 'й': 10,

            'к': 11, 'л': 12, 'м': 13, 'н': 14, 'о': 15,

            'п': 16, 'р': 17, 'с': 18, 'т': 19, 'у': 20,

            'ф': 21, 'х': 22, 'ц': 23, 'ч': 24, 'ш': 25,

            'щ': 26, 'ъ': 27, 'ы': 28, 'ь': 29, 'э': 30,

            'ю': 31, 'я': 32

            }

def IsPrime(n):

    d = 2

    while n % d != 0:

        d += 1

    return d == n

def modInverse(e, el):

    e = e % el

    for x in range(1, el):

        if ((e \* x) % el == 1):

            return x

    return 1

def is\_prime(num, test\_count):

    if num == 1:

        return False

    if test\_count >= num:

        test\_count = num - 1

    for x in range(test\_count):

        val = random.randint(1, num - 1)

        if pow(val, num-1, num) != 1:

            return False

    return True

def gen\_prime(n):

    found\_prime = False

    while not found\_prime:

        p = random.randint(2\*\*(n-1), 2\*\*n)

        if is\_prime(p, 1000):

            return p

def hash\_value(mod, alpha\_code\_msg):

    i = 0

    hashing\_value = 1

    while i < len(alpha\_code\_msg):

        hashing\_value = (((hashing\_value-1) + int(alpha\_code\_msg[i]))\*\*2) % mod

        i += 1

    return hashing\_value

def egcipher(clearText):

    p = gen\_prime(10)

    print("P =", p)

    g = random.randint(2, p-1)

    print("G =", g)

    x = random.randint(2, p-2)

    y = (g\*\*x) % p

    print("Открытый ключ(Y)={}, Секретный ключ(X)={}".format(y, x))

    msg = clearText

    msg\_list = list(msg)

    alpha\_code\_msg = list()

    for i in range(len(msg\_list)):

        alpha\_code\_msg.append(int(alphavit.get(msg\_list[i])))

    print("Длина исходного сообщения {} символов".format(len(alpha\_code\_msg)))

    hash\_code\_msg = hash\_value(p, alpha\_code\_msg)

    print("Хэш сообщения:= {}".format(hash\_code\_msg))

    k = 1

    while True:

        k = random.randint(1, p-2)

        if gcd(k, p-1) == 1:

            print("K =", k)

            break

    a = (g\*\*k) % p

    b = (hash\_code\_msg - (x\*a)) % (p-1)

    print("Значение подписи:S={},{}".format(a, b))

    b = modInverse(k, p-1) \* ((hash\_code\_msg - (x \* a)) % (p-1))

    check\_hash\_value = hash\_value(p, alpha\_code\_msg)

    a\_1 = ((y\*\*a) \* (a\*\*b)) % p

    print("A1={}".format(a\_1))

    a\_2 = (g\*\*check\_hash\_value) % p

    print("A2={}".format(a\_2))

    if a\_1 == a\_2:

        print("Подпись верна\n")

    else:

        print("Подпись неверна")

print('ЭЦП Elgamal:')

print('КОРОТКИЙ ТЕКСТ:')

egcipher(input\_for\_cipher\_short())

print('ДЛИННЫЙ ТЕКСТ:')

egcipher(input\_for\_cipher\_long())

**Тестирование:**

/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab09\_25\_elgamal.py

ЭЦП Elgamal:

КОРОТКИЙ ТЕКСТ:

P = 907

G = 875

Открытый ключ(Y)=665, Секретный ключ(X)=617

Длина исходного сообщения 39 символов

Хэш сообщения:= 376

K = 769

Значение подписи:S=550,776

A1=194

A2=194

Подпись верна

ДЛИННЫЙ ТЕКСТ:

P = 587

G = 410

Открытый ключ(Y)=109, Секретный ключ(X)=161

Длина исходного сообщения 1087 символов

Хэш сообщения:= 423

K = 35

Значение подписи:S=226,369

A1=102

A2=102

Подпись верна

**Блок-схема:**

# Блок J: СТАНДАРТЫ ЦИФРОВЫХ ПОДПИСЕЙ

## ГОСТ Р 34.10-94

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

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

а - любое число, большее 1 и меньшее (р-1), причем такое, что аq mod p=1;

х - некоторое число, меньшее q;

у = аx mod р.

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

**Код программы:**

from base import alphabet, input\_for\_cipher\_short, input\_for\_cipher\_long, output\_from\_decrypted

alphavit = {'а': 0, 'б': 1, 'в': 2, 'г': 3, 'д': 4,

            'е': 5, 'ё': 6, 'ж': 7, 'з': 8, 'и': 9, 'й': 10,

            'к': 11, 'л': 12, 'м': 13, 'н': 14, 'о': 15,

            'п': 16, 'р': 17, 'с': 18, 'т': 19, 'у': 20,

            'ф': 21, 'х': 22, 'ц': 23, 'ч': 24, 'ш': 25,

            'щ': 26, 'ъ': 27, 'ы': 28, 'ь': 29, 'э': 30,

            'ю': 31, 'я': 32

            }

def ciphergostd(clearText):

    array = []

    flag = False

    for s in range(50, 1000):

        for i in range(2, s):

            if s % i == 0:

                flag = True

                break

        if flag == False:

            array.append(s)

        flag = False

    p = 31

    print("p = ", p)

    q = 5

    print("q = ", q)

    a = 2

    print("a =", a)

    array2 = []

    flag2 = False

    for s in range(2, q):

        for i in range(2, s):

            if s % i == 0:

                flag2 = True

                break

        if flag2 == False:

            array2.append(s)

        flag2 = False

    x = 3

    print("x = ", x)

    y = a\*\*x % p

    k = 4

    print("k = ", k)

    r = (a\*\*k % p) % q

    msg = clearText

    msg\_list = list(msg)

    alpha\_code\_msg = list()

    for i in range(len(msg\_list)):

        alpha\_code\_msg.append(int(alphavit.get(msg\_list[i])))

    print("Длина исходного сообщения {} символов".format(len(alpha\_code\_msg)))

    hash\_code\_msg = hash\_value(p, alpha\_code\_msg)

    print("Хэш сообщения:= {}".format(hash\_code\_msg))

    s = (x\*r+k\*hash\_code\_msg) % q

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

    v = (hash\_code\_msg\*\*(q-2)) % q

    z1 = s\*v % q

    z2 = ((q-r)\*v) % q

    u = (((a\*\*z1)\*(y\*\*z2)) % p) % q

    print(r, " = ", u)

    if u == r:

        print("r = u, следовательно:")

        print("Подпись верна\n")

    else:

        print("Подпись неверна")

def hash\_value(n, alpha\_code):

    i = 0

    hash = 1

    while i < len(alpha\_code):

        hash = (((hash-1) + int(alpha\_code[i]))\*\*2) % n

        i += 1

    return hash

print('ГОСТ Р 34.10-94:')

print('КОРОТКИЙ ТЕКСТ:')

ciphergostd(input\_for\_cipher\_short())

print('ДЛИННЫЙ ТЕКСТ:')

ciphergostd(input\_for\_cipher\_long())

**Тестирование:**

/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab10\_26\_gost94.py

ГОСТ Р 34.10-94:

КОРОТКИЙ ТЕКСТ:

p = 31

q = 5

a = 2

x = 3

k = 4

Длина исходного сообщения 39 символов

Хэш сообщения:= 28

Цифровая подпись = 1 , 0

1 = 1

r = u, следовательно:

Подпись верна

ДЛИННЫЙ ТЕКСТ:

p = 31

q = 5

a = 2

x = 3

k = 4

Длина исходного сообщения 1087 символов

Хэш сообщения:= 20

Цифровая подпись = 1 , 3

1 = 1

r = u, следовательно:

Подпись верна

**Блок-схема:**

## ГОСТ Р 34.10-2012

Для сообщества пользователей выбирается общая эллиптическая кривая Ep(a, b) и точка G на ней, такая, что G, [2]G, [3]G, …, [q]G суть различные точки, и [q]G = О для некоторого простого числа q (длина числа q равна 256 бит). Каждый пользователь U выбирает случайное число xu (секрет ный ключ), 0 <хu < q, и вычисляет точку на кривой Yu = [xu]G (открытый ключ). Параметры кривой и список открытых ключей передаются всем пользователям.

Чтобы подписать сообщение , пользователь А делает следующее:

1. Вычисляет значение хеш-функции сообщения h = h();
2. Выбирает случайно число k, 0 < k < q;
3. Вычисляет Р = [k]G = (х, у);
4. Вычисляет r = х mod q (при r = 0 возвращается к шагу 2);
5. Вычисляет s = (kh + rха) mod q (при s = 0 возвращается к шагу 2);
6. Подписывает сообщение парой чисел (r, s).

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

1. Вычисляет h = h();
2. Убеждается, что 0 < r, s < q;
3. Вычисляет u1 = s · h-1 mod q и u2 = – r · h-1 mod q;
4. Вычисляет композицию точек на кривой Р = [u1]G + [u2]YA = (х, у) и, если Р = О, отвергает подпись;
5. Если x mod q = r, принимает подпись, в противном случае отвергает ее.

**Код программы:**

import random

import collections

from base import alphabet, input\_for\_cipher\_short, input\_for\_cipher\_long, output\_from\_decrypted

alphabet\_lower = {'а': 0, 'б': 1, 'в': 2, 'г': 3, 'д': 4,

            'е': 5, 'ё': 6, 'ж': 7, 'з': 8, 'и': 9, 'й': 10,

            'к': 11, 'л': 12, 'м': 13, 'н': 14, 'о': 15,

            'п': 16, 'р': 17, 'с': 18, 'т': 19, 'у': 20,

            'ф': 21, 'х': 22, 'ц': 23, 'ч': 24, 'ш': 25,

            'щ': 26, 'ъ': 27, 'ы': 28, 'ь': 29, 'э': 30,

            'ю': 31, 'я': 32

            }

class Point:

    def \_\_init\_\_(self, x\_init, y\_init):

        self.x = x\_init

        self.y = y\_init

    def shift(self, x, y):

        self.x += x

        self.y += y

    def \_\_repr\_\_(self):

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

x\_1 = 0

y\_1 = 0

EllipticCurve = collections.namedtuple(

    'EllipticCurve', 'name p q\_mod a b q g n h')

curve = EllipticCurve(

    'secp256k1',

    p=0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffefffffc2f,

    q\_mod=0xfffffffffefffffffffcfffffffffffcfffffffffffffffffffffffefffffc2f,

    a=7,

    b=11,

    g=(0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798,

       0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8),

    q=(0xA0434D9E47F3C86235477C7B1AE6AE5D3442D49B1943C2B752A68E2A47E247C7,

       0x893ABA425419BC27A3B6C7E693A24C696F794C2ED877A1593CBEE53B037368D7),

    n=0xfffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141,

    h=1,

)

def ciphergosto(clearText):

    msg = clearText

    msg\_list = list(msg)

    alpha\_code\_msg = list()

    for i in range(len(msg\_list)):

        alpha\_code\_msg.append(int(alphabet\_lower.get(msg\_list[i])))

    print("Длина исходного сообщения {} символов".format(len(alpha\_code\_msg)))

    print("Q mod", int(curve.q\_mod))

    print("P mod", int(curve.p))

    hash\_code\_msg = hash\_value(curve.p, alpha\_code\_msg)

    print("Хэш сообщения:={}".format(hash\_code\_msg))

    e = hash\_code\_msg % curve.q\_mod

    print("E={}".format(e))

    k = random.randint(1, curve.q\_mod)

    print("K={}".format(k))

    d = 10

    print("D={}".format(d))

    x, y = scalar\_mult(k, curve.g)

    point\_c = Point(x, y)

    print("Point\_C={}".format(point\_c))

    r = point\_c.x % curve.q\_mod

    print("R={}".format(r))

    s = (r\*curve.p + k\*e) % curve.q\_mod

    print("S={}".format(s))

    v = inverse\_mod(e, curve.p)

    print("V={}".format(v))

    z1 = (s\*v) % curve.q\_mod

    z2 = ((curve.p-r)\*v) % curve.q\_mod

    x\_1, y\_1 = scalar\_mult(d, curve.g)

    print("Point\_Q=( x={}, y={} )".format(x\_1, y\_1))

    point\_c\_new = Point(x, y)

    x, y = point\_add(scalar\_mult(z1, curve.g),

                     scalar\_mult(z2, curve.q))

    r\_1 = point\_c\_new.x % curve.q\_mod

    print("R\_new={}".format(r\_1))

    if r == r\_1:

        print("Подпись прошла проверку!\n")

    else:

        print("Ошибка проверки!")

def hash\_value(mod, alpha\_code\_msg):

    i = 0

    hashing\_value = 1

    while i < len(alpha\_code\_msg):

        hashing\_value = (

            ((hashing\_value-1) + int(alpha\_code\_msg[i]))\*\*2) % curve.p

        i += 1

    return hashing\_value

def is\_on\_curve(point):

    if point is None:

        return True

    x, y = point

    return (y \* y - x \* x \* x - curve.a \* x - curve.b) % curve.p == 0

def point\_neg(point):

    if point is None:

        return None

    x, y = point

    result = (x, -y % curve.p)

    return result

def inverse\_mod(k, p):

    if k == 0:

        raise ZeroDivisionError('деление на 0')

    if k < 0:

        return p - inverse\_mod(-k, p)

    s, old\_s = 0, 1

    t, old\_t = 1, 0

    r, old\_r = p, k

    while r != 0:

        quotient = old\_r // r

        old\_r, r = r, old\_r - quotient \* r

        old\_s, s = s, old\_s - quotient \* s

        old\_t, t = t, old\_t - quotient \* t

    gcd, x, y = old\_r, old\_s, old\_t

    assert gcd == 1

    assert (k \* x) % p == 1

    return x % p

def point\_add(point1, point2):

    if point1 is None:

        return point2

    if point2 is None:

        return point1

    x1, y1 = point1

    x2, y2 = point2

    if x1 == x2 and y1 != y2:

        return None

    if x1 == x2:

        m = (3 \* x1 \* x1 + curve.a) \* inverse\_mod(2 \* y1, curve.p)

    else:

        m = (y1 - y2) \* inverse\_mod(x1 - x2, curve.p)

    x3 = m \* m - x1 - x2

    y3 = y1 + m \* (x3 - x1)

    result = (x3 % curve.p,

              -y3 % curve.p)

    return result

def scalar\_mult(k, point):

    if k % curve.n == 0 or point is None:

        return None

    if k < 0:

        return scalar\_mult(-k, point\_neg(point))

    result = None

    addend = point

    while k:

        if k & 1:

            result = point\_add(result, addend)

        addend = point\_add(addend, addend)

        k >>= 1

    return result

print('ГОСТ Р 34.10-2012:')

print('КОРОТКИЙ ТЕКСТ:')

ciphergosto(input\_for\_cipher\_short())

print('ДЛИННЫЙ ТЕКСТ:')

ciphergosto(input\_for\_cipher\_long())

**Тестирование:**

/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab10\_27\_gost2012.py

ГОСТ Р 34.10-2012:

КОРОТКИЙ ТЕКСТ:

Длина исходного сообщения 39 символов

Q mod 115792089237210883131902140479076077470404524942491262870694982560773809634351

P mod 115792089237316195423570985008687907853269984665640564039457584007908834671663

Хэш сообщения:=100465605960392405545326615989120765479067559508215715274908601668339432312127

E=100465605960392405545326615989120765479067559508215715274908601668339432312127

K=43019610968597533575694927402782043341666815463049898570394327923006212119082

D=10

Point\_C=( x=80553680982383746563554500880210053620115828941187278030733140405035128490425, y=74824895621507411931400280064663892593331393722726527632339907349142076474785)

R=80553680982383746563554500880210053620115828941187278030733140405035128490425

S=35581294126978857713435754913480675793101545360322986846601181291383678561299

V=20529166449300022691683847027495261733238752076834297463717225331510851960291

Point\_Q=( x=109805586211166206629432866892583231117554510260596600142888290125507993067118, y=51243083235504058321191534323736250822297443681753984114121156474938550647252 )

R\_new=80553680982383746563554500880210053620115828941187278030733140405035128490425

Подпись прошла проверку!

ДЛИННЫЙ ТЕКСТ:

Длина исходного сообщения 1087 символов

Q mod 115792089237210883131902140479076077470404524942491262870694982560773809634351

P mod 115792089237316195423570985008687907853269984665640564039457584007908834671663

Хэш сообщения:=44642821119098115386834658767856322616357022834257596733930388099678580390171

E=44642821119098115386834658767856322616357022834257596733930388099678580390171

K=48431864128040811700370753657562279379539026068468713299770002081598293602354

D=10

Point\_C=( x=44800459081610348166673508108385128878705452232210702982677761178660890566077, y=13195091569128410442209436744524910967401329637352914815201625579914336769926)

R=44800459081610348166673508108385128878705452232210702982677761178660890566077

S=110804626836197049152945897279988234683557906819154210898078501121592663046804

V=24078682445532384626955465808779465306725321854082666482660247247597808478951

Point\_Q=( x=109805586211166206629432866892583231117554510260596600142888290125507993067118, y=51243083235504058321191534323736250822297443681753984114121156474938550647252 )

R\_new=44800459081610348166673508108385128878705452232210702982677761178660890566077

Подпись прошла проверку!

**Блок-схема:**

# Блок K: Обмен ключами

## ОБМЕН КЛЮЧАМИ ПО ДИФФИ-ХЕЛЛМАНУ

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

1. Определить секретные ключи пользователей КА и КВ.

2. Для этого каждый пользователь независимо выбирает случайные числа из интервала (2,..., n-1).

3. Вычислить открытые ключи пользователей YA и YB: Y=aK mod n

4. Обменяться ключами YA и YB по открытому каналу связи.

5. Независимо определить общий секретный ключ К: KA=YKA mod n KB=YKB mod n.

KA = KB = K

**Код программы:**

a = int(input("Введите число а: "))

n = int(input("Введите число n, n должно быть больше a: "))

ka = int(input("Введите число ka: "))

Ya = a\*\*ka % n

print ("Ваш Ya = ", Ya)

Yb = int(input("Введите число Yb, которое прислал собеседник: "))

K = (a\*\*(Ya\*Yb))%n

print ("Ваш общий ключ: ", K)

**Тестирование:**

/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab11\_28\_dh.py

Введите число а: 10

Введите число n, n должно быть больше a: 30

Введите число ka: 4

Ваш Ya = 10

Введите число Yb, которое прислал собеседник: 20

Ваш общий ключ: 10

**Блок-схема:**