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

ЛАБОРАТОРНЫЕ РАБОТЫ ПО ПРЕДМЕТУ

«Программирование криптографических алгоритмов»

Выполнил:
Барышников С.С. гр. 191-351
Преподаватель:
Бутакова Н.Г.

Москва 2021 г.

Содержание

Аннотация

Среда программирования: Visual Studio Code

Язык программирования: Python 3

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

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

Текст для проверки работы: Вот пример статьи на тысячу символов. Это достаточно маленький текст, оптимально подходящий для карточек товаров в интернет или магазинах или для небольших информационных публикаций. В таком тексте редко бывает более двух или трёх абзацев и обычно один подзаголовок. Но можно и без него. На тысячу символов рекомендовано использовать один или два ключа и одну картину. Текст на тысячу символов это сколько примерно слов? Статистика показывает, что тысяча включает в себя 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

1. Шифр простой замены АТБАШ

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

Код программы:

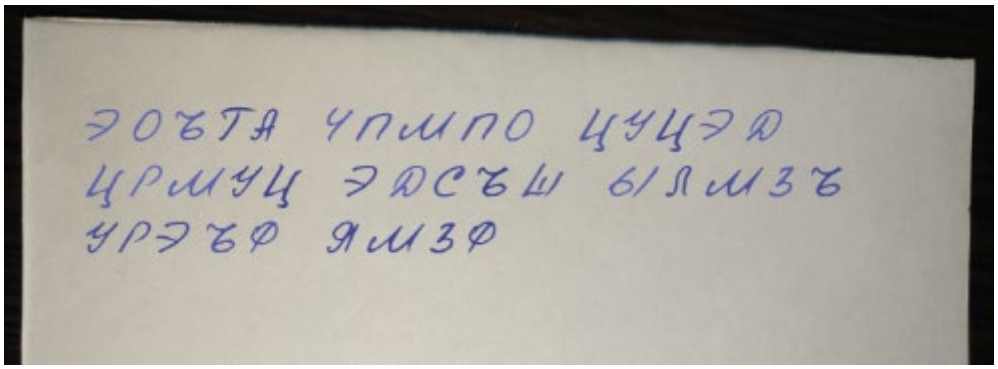
```
from base import alphabet, input_for_cipher_short, input_for_cipher_long,
output_from_decrypted

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

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

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

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

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

Тестирование:

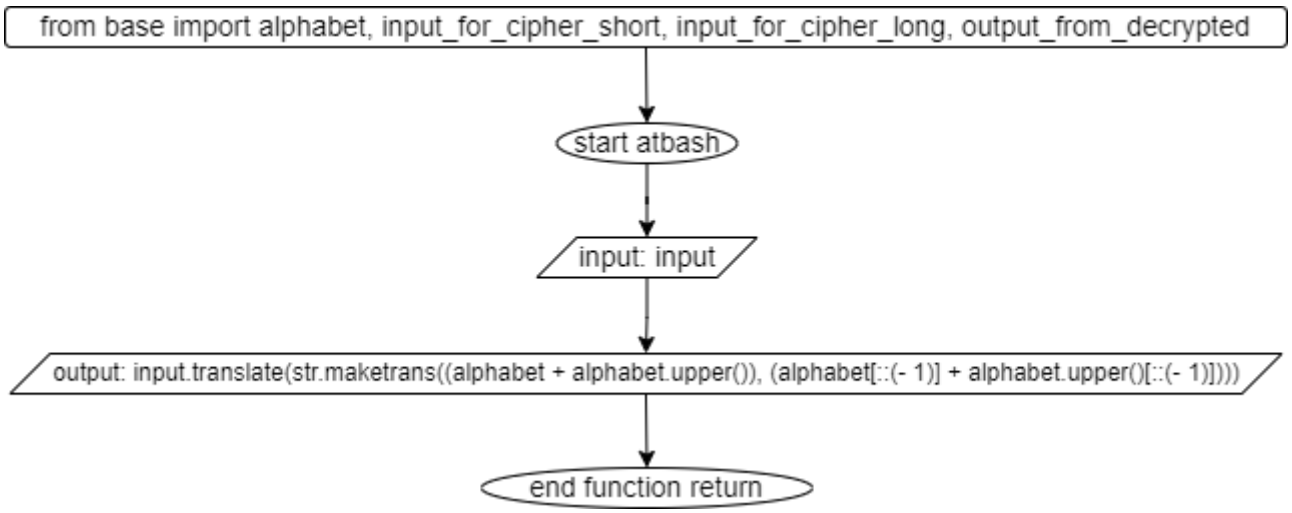
```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab01_1_atbash.py
ШИФР АТБАШ:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
эоътачпмпоцуцэдцрмуцэдсъшылмзъурэъфямзф

Расшифрованный текст:
время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
эрмпоцтъонмямгцсямдназлнцтэрурэмзфвмрырнмямрзсртяуъсгфцхмъфнмчпмрпмцтяугсрпрый
рыаёцхыуафяомрзъфмрэяорээцсмъосъмцуцтяьячцсяйцуцыуасъюругжцйцскротяицрссдйплюу
цфяицхмзфэмяфртмъфнмъоъыфрюдэяъмюруъъыэлйцуцмощйяючяиъэцрюдзсррыцспрычяьрурэрф
мзфсртршсрцюъчсъьрмзфсямдназлнцтэрурэоъфртъсырэясрцнпругчрэямгрыцсцуцыэяфубзяц
рыслфяомцслмзфмъфнмсямдназлнцтэрурэвмрнфругфрпоцтъосрнурэмзфнмямцнмцфяпрфячдэя
ъмчпмзмрмдназяэфубзяъмэнъюанмрпамгыънамцуцыэънмцнурэноъысъхэъуцзцсдмзфсрчпмъну
цчурлпрмоъюуамгпоъыурьятцчпмнрбчятццыольцтцзянматцоъзцсярыцсцуцыэянцтэруячпммр
```

```
фруцзънмэрнурэсъцчтъссрэрчоянмяъммзфэфрпцояхмъонфрхыъамъугсрнмцпоцсамрнзцмямгм
дназцнпорюъуятццуцюъчмзфлзъмпорюъурэлэъуцзцэяъмрюеътмъфнмяпоцтъосрсянмрцуцыэън
мцнцтэрурэцтъссрнмругфроячтдоячыъуаътнурэянэрюрысдтпорнмояснмэртмзфнзцмямгпорю
ъудчяфячзцфцсъубюамчпммяффяфвмрплнмръттънмрмзфрысяфрсъфрмродъкцотдцюцошцэцыамнп
ояэъыуцэдтнмяэцмгнмрцтрнмгчямдназлнцтэрурэнпорюъуятцчпмнзцмяапрнуъысцъэяшсдтву
ътъсмртфязънмэъссрьрэрнпоцамцамзфнрьуянцмънгчпмзцмямгнуцмсдхмъфнмюъчъыцсрьрпор
плнфячпмсцфмрсъюлыъммзфсрюругжцснмэлслшсяиъсячямдназлчсяфрэюъчпорюъурэмзф

Расшифрованный текст:
вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходящ
ийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпубликац
ий.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.номож
ноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднукартин
у.текстнатысячусимволовэтосколькопримернослов.статистикапоказывает,чтотысячавк
лючаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблятьпредл
огами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеизменнов
озрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиилибез.учет
пробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименностолькоразмы
разделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,таккакэтоп
устоеместо.однаконекоторыефирмыибирживидятсправедливымставитьстоимостьзатысячу
символовспробелами,считаяпоследниеважнымэлементомкачественноговосприятия.согла
ситесь,читатьслитныйтекстбезединогопропуска,никтонебудет.нобольшинствунужнацен
азатысячузнаковбезпробелов.
```

Карточка:



Блок-схема:



2. ШИФР ЦЕЗАРЯ

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

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

Код программы:

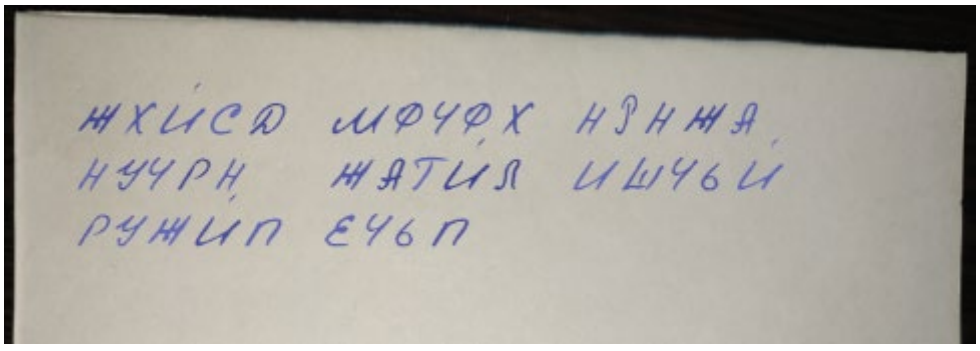
```
from base import alphabet, input_for_cipher_short, input_for_cipher_long,
output_from_decrypted
import random

key = 5

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

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

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

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

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

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

Тестирование:

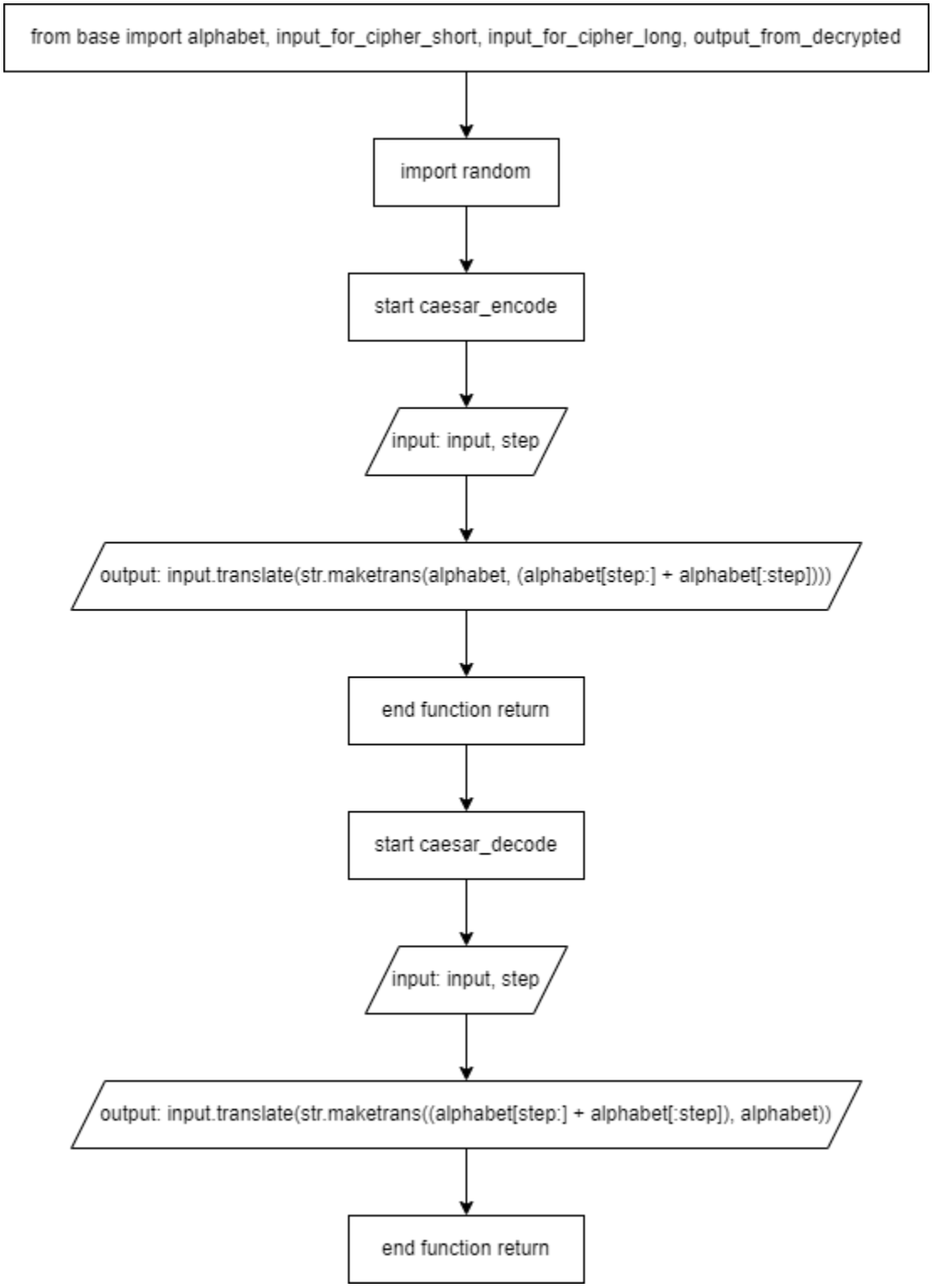
```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab01_2_caesar.py

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

```
Зашифрованный текст:
жхйсдмфчфхнрнжанучрнжатйлишчьйружйпечьп

Расшифрованный текст:
время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
жучфхнсйхцчечбнтечацдьшцнсжуружчьпвчуиуцчечуьтусерйтбпночйпцчмфчуфчнсербтуфуиъ
уидюноирдпехчуьйпчужехужжнтчйхтйчнрнсеземнтеънрнирдтйёурбэнънтщухсеынуттаъфшёр
нпеыночьпжчепусчйпцчйхйипуёажейчёурййижшънрнчхкъеёмеыйжнуёаьтууинтфуимезуружуп
чьптусултунёймтйзучьптечацдьшцнсжуружхйпусйтиужетунцфурбмужечбуинтнрнижепргьен
уитшпехчнтшчьпчйпцчтечацдьшцнсжуружвчуцпурбпуфхнсйхтуцружчьпцчечнцчнпефупемаже
йчмфчьчучацдьежпргьейчжцйёдццчуфдчбийцдчнрнижйцчнцружцхйитйожйрньнтачьптумфчйцр
нмрушфучхйёрдчбфхйирузеснмфчцугмесннихшзнсньецчдснхйьнтеуинтнрнижецнсжуремфччу
пурньйцчжуцружтйнмсйттужумхецчейччьпжпуфнхеочйхцпуоийдчйрбтуцчнфхнтдчуцьнчечбч
ацдьнцфхуёйресннрнёймчьпшьйчфхуёйружшжйрньнжейчуёяйсчйпцчефхнсйхтутецчунрнижйц
чнцнсжуружнсйттуцчурбпухемсахемийрдйсцружецжуёуитасфхуцчхетцчжусчьпцьнчечбфхуё
йрамепемьнпнтйргёдчмфччеппепвчуфшцчуйсйцчучьпуитепутйпучухайщнхсанёнхлнжнидчцф
хежйирнжасцчежнчбцчунсуцчбмечацдьшцнсжуружцфхуёйреснмфчцьнчедфуцрйитнйжелтасвр
йсйтчуспеьйцчжйттузужуцфхндчндчьпцузрецнчйцбмфчьнчечбцрнчтаочйпцчёймйинтузуфху
фшцпемфчтнпчутйёшийччьптуёурбэнтцчжштшлтеыйтемечацдьшмтепужёймфхуёйружчьп

Расшифрованный текст:
вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходящ
ийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпубликац
ий.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.номож
ноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднукартин
у.текстнатысячусимволовэтосколькопримернослов.статистикапоказывает,чтотысячавк
лючаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблятьпредл
огами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеизменнов
озрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиилибез.учет
пробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименностолькоразмы
разделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,таккакэтоп
устоеместо.однаконекоторыефирмыибиржividятсправедливымставитьстоимостьзатысячу
символовспробелами,считаяпоследниеважнымэлементомкачественноговосприятия.согла
ситесь,читатьслитныйтекстбезединогопропуска,никтонебудет.небольшинствунужнацен
азатысячузнаковбезпробелов.
```

Карточка:



Блок-схема:


from base import alphabet, input_for_cipher_short, input_for_cipher_long, output_from_decrypted
import random
start caesar_encode
input: input, step
output: input.translate(str.maketrans(alphabet, (alphabet[step:] + alphabet[:step])))
end function return
start caesar_decode
input: input, step
output: input.translate(str.maketrans((alphabet[step:] + alphabet[:step]), alphabet))
end function return

3. Квадрат Полибия

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

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

Код программы:

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

hard_dictionary = {"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:

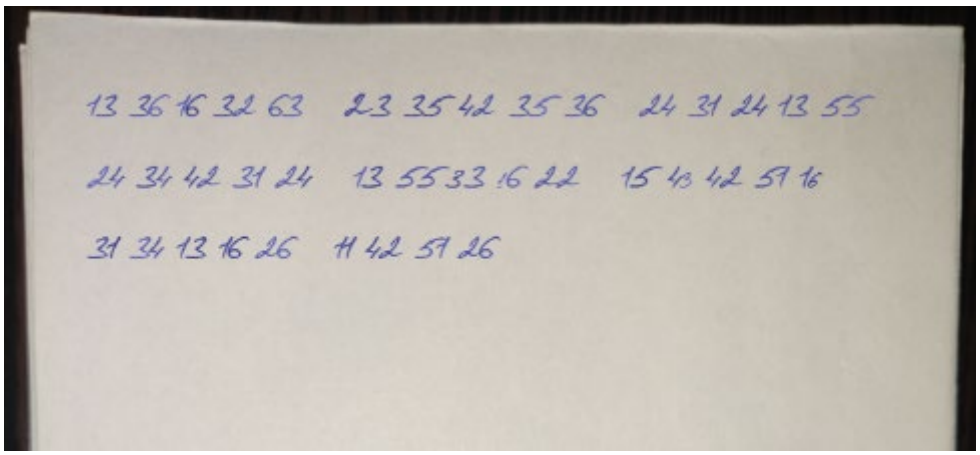
```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab01_3_square.py

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:
133442353624321636414211425624331142554163514341243213343134134251266142341534
414211423451333432113116335626242542162641422335423435422432113156333435341545
341563532425153163261136423451162642341311363413132433421636331642243124321114
112324331145243124153163331612343156522445243344343632114624343333554535431231
242611462425425126134211263432421626414216361615263412551311164212343116161513
434524312442362145111223114616132434125551333434152433353415231114343134133426
425126333432342233342412162333161434425126331142554163514341243213343134133616
263432163315341311333424413534315623341311425634152433243124151311263162511124
341533432611364224334342512642162641423311425541635143412432133431341361423441
263431562634353624321636333441313413425126414211422441422426113534261123551311
```

```
164223354251423442554163511113263162511116421341161263414234356342561516416342
243124151316414224413134134136161533162513163124512433554251263334233542164131
242331344335344236161231634256353616153134141132242335424134622311322424153643
142432245111414263322436165124331134152433243124151311412432133431112335424234
263431245116414213344131341333162423321633333413342336114142111642425126132634
352436112542163641263425151663421631563334414224353624336342344151244211425642
554163512441353634121631113224243124121623425126435116423536341216313413431316
312451241311164234125416324216264142113536243216363334331141423424312415131641
422441243213343134132432163333344142343156263436112332553611231516316316324131
341311411334123415335532353634414236113341421334324251264151244211425635363412
163155231126112351242624331631621263422335424211262611266142343543414234163216
414234425126341533112634331626344234365516442436325524122436222413241563424135
361113161531241355324142111324425641423424323441425623114255416351434124321334
313413413536341216311132242335424151244211633534413116153324161311223355326131
163216334234322611511641421316333334143413344135362463422463425126413414311141
244216415623354251244211425641312442335525421626414212162316152433341434353634
354341261123354233242642343316124315164242512633341234315652243341421343334322
331146163311231142554163514323331126341312162335363412163134134251 26
```

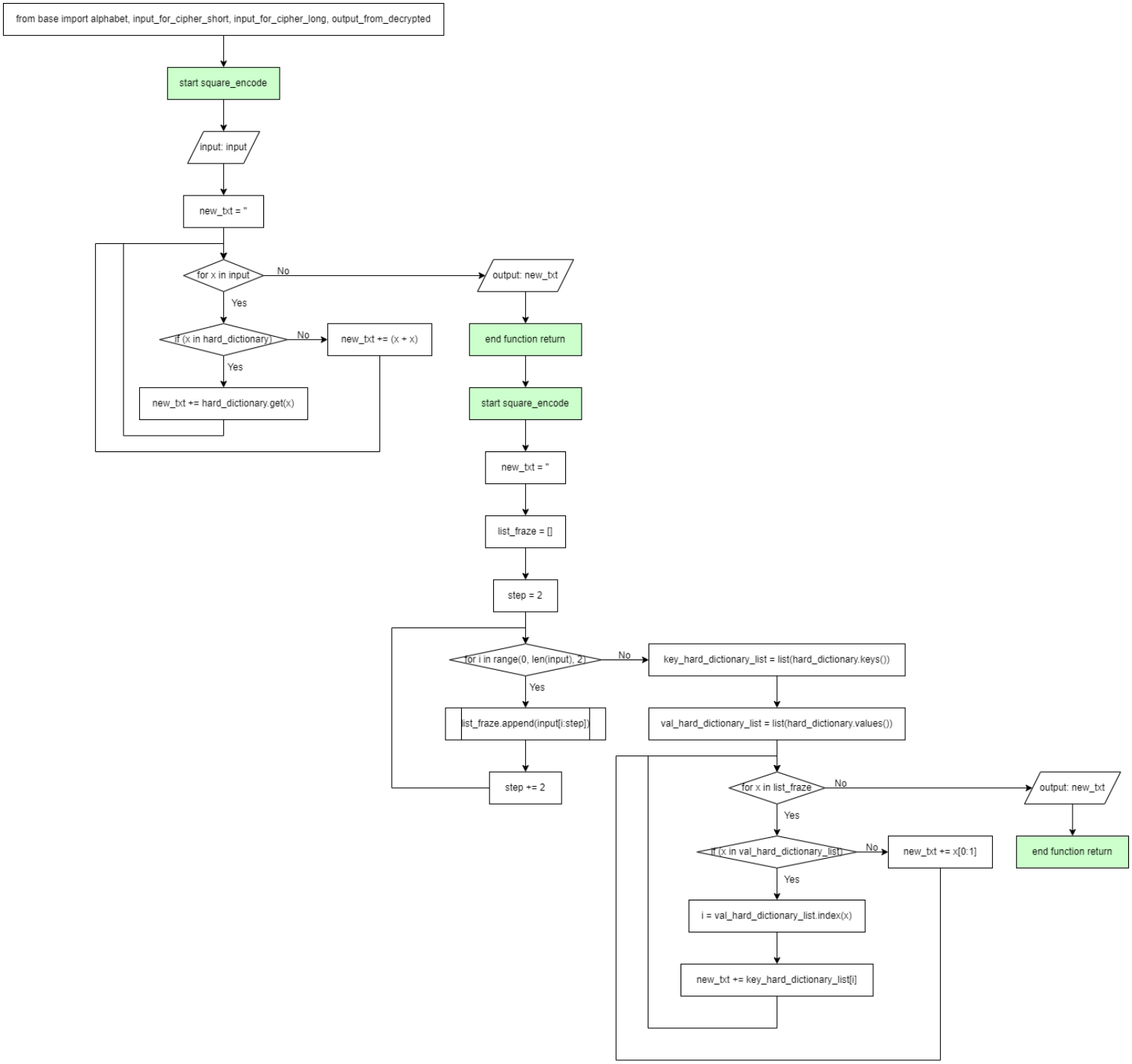
Расшифрованный текст:

вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходящийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпубликаций.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.номожноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднукартинку.текстнатысячусимволовэтосколькопримернослов.статистикапоказывает,чтотысячавключаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблятьпредлогами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеизменновозрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиилибез.учетпробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименностолькоразмыразделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,таккакэтопустоеместо.однаконекоторыефирмыибиржевидятсправедливымставитьстоимостьзатысячусимволовспробелами,считаяпоследниеважнымэлементомкачественноговосприятия.согласитесь,читатьслитныйтекстбезединогопропуска,никтонебудет.небольшинствунужнаценазатысячузнаковбезпробелов.

Карточка:



Блок-схема:


from base import alphabet, input_for_cipher_short, input_for_cipher_long, output_from_decrypted
start square_encode
input: input
new_txt = ''
for x in input
No
Yes
output: new_txt
if (x in hard_dictionary)
No
new_txt += (x + x)
Yes
new_txt += hard_dictionary.get(x)
end function return
start square_encode
new_txt = ''
list_fraze = []
step = 2
for i in range(0, len(input), 2)
No
Yes
key_hard_dictionary_list = list(hard_dictionary.keys())
list_fraze.append(input[i:step])
val_hard_dictionary_list = list(hard_dictionary.values())
step += 2
for x in list_fraze
No
Yes
output: new_txt
if (x in val_hard_dictionary_list)
No
new_txt += x[0:1]
Yes
end function return
i = val_hard_dictionary_list.index(x)
new_txt += key_hard_dictionary_list[i]

Блок В: ШИФРЫ МНОГОЗНАЧНОЙ ЗАМЕНЫ

4. Шифр Тритемия

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

Преимущество этого метода шифрования по сравнению с методом Альберти состоит в том, что с каждой буквой задействуется новый алфавит. Альберти менял алфавиты лишь по-сле 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(
```

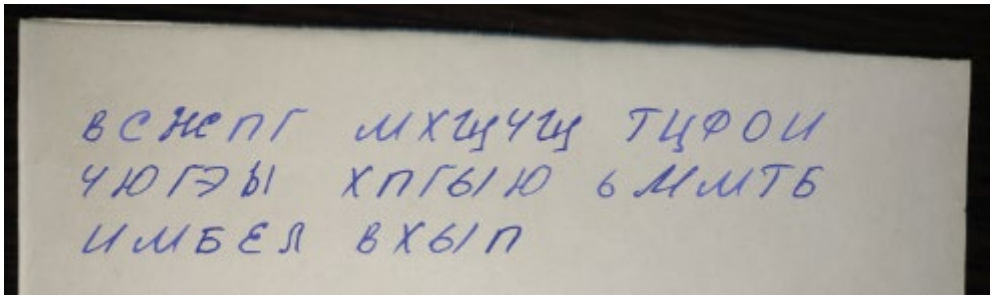
```
    input_for_cipher_short())))}

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

Расшифрованный текст:
{output_from_decrypted(trithemius_decode(trithemius_encode(
    input_for_cipher_long())))}
''')
```

Тестирование:

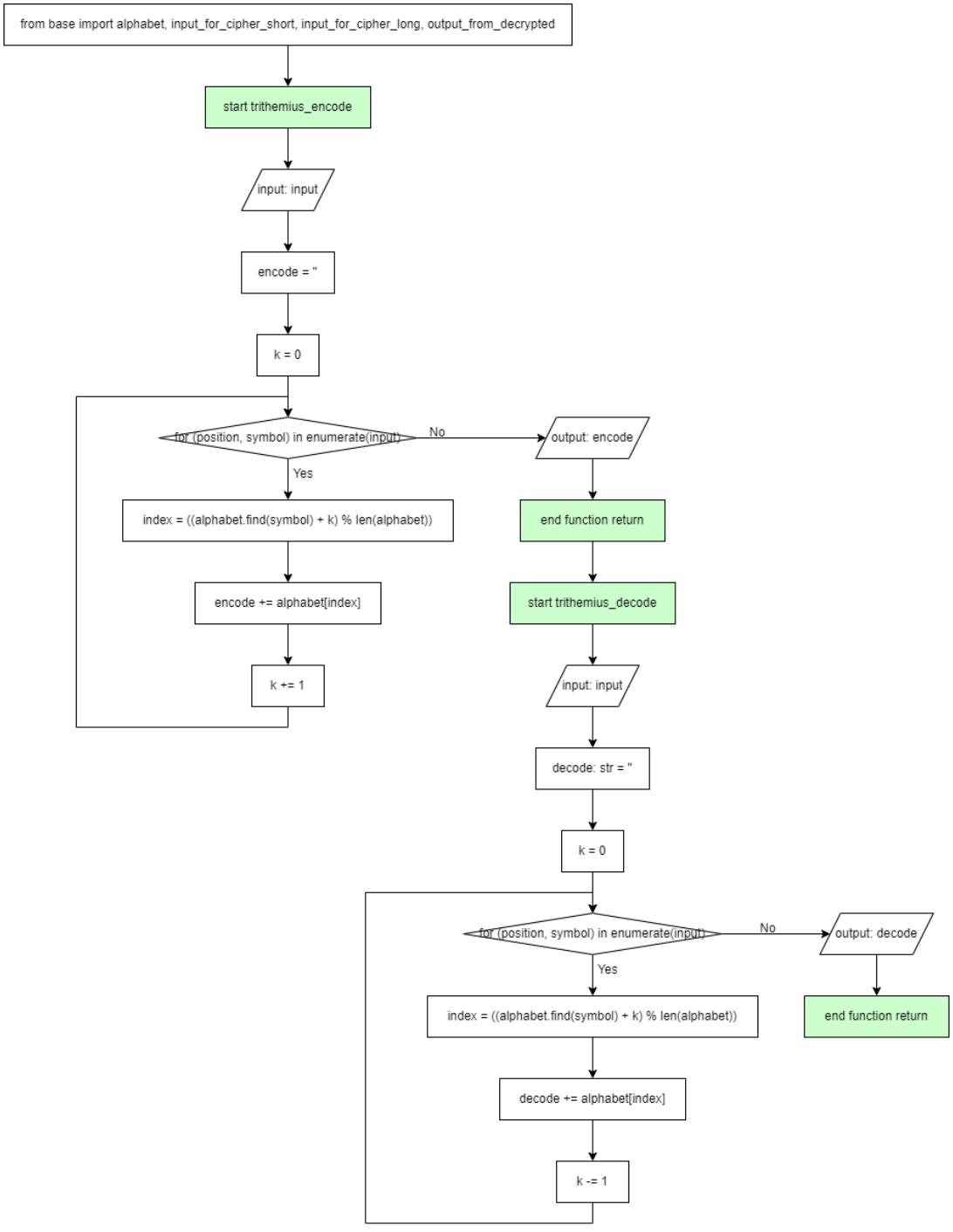
```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-
1/lab02_4_trithemius.py

Шифр Тритемия:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
всжпгмхщчщтцфоичюгэыхпгыюьммтбимбелвхып

Расшифрованный текст:
время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
впфтфнтлшъькюицьпгмдтлизаеыижкярцкюфсзучщзышвщыъоыхяоюэяиьгкмглпмотйогпбухччна
ърмзшъхютяхжйжряёолаярпдемтшлшцоэфшцыпусъьвхладвюжыкгаомюымъофьчъчлгцээюмзгзцв
агщрдёпхйвувнтсшлтъьпьссщюсмфущзчёдюяяюрузлйфуйъёзпиапнхьпкзябвшюджжэвыялйнвпм
хыпухфчршъхоучюцхвжмбешлхмыфсриндспузчмушчръсэсрябьёеегфбиэьпъндйплпнйизухигмц
эуюеэуезяеллсёовиртовхяцеюыътчныщэсндбеядвугзйлейгпнпуотжешъиьуэяцщпааэуьршчэл
вкофрнтьувыезсужбкряпафсрдёгехйфэямыпжкиедзхчошучлььссьфъучяэмяеулсёйлёотёуомм
схышшэьсоъогнвдщщвёыщжддмррояйгрнокшьмушхеобгряьеънаёщияекжкиедгхнтицфйтыяаэъя
эыкщжищжкёйюкгззнжрсузхпщйьюмтбвзьфюгеязшгмамоинйежвцйсгхчыэфьючэпдбелюичкхмцн
ьхзсртсейсжфстцфнтцвзьёкщзёзжъяжкясбемкъмжёлъчкерщауъвдтгеюгьижиьорнимкжёйчщр
ътчныщэсщючбвдзйжецёкнюжмьтгликтжнцьчыпобтаувшгсдзйимонюмсудсрчэсщатэляйюаятчя
пэцвшбсджёлдлягвцмхщъифхлчкбюаедёгъгьйлинузмгнбмссйрхъчъёрбцяьлъуьеьящбэщннмоя
вёёомжбущыймфяйавяъявёзтяшлхмыфсриндспузшчщшлршнышшбеелюиччиимзввмиёдгктбуефоч
ръаэьысйшёзшьёжиюкямрпсквцнещяуыщошнашгцдпьеиоблъншойзтоэмцйршъйотррцьюуавдгей
игщвкокжйтппзешйлъыбхщыоэымйыбёзшкёмбиьугмаивхяцеюырчкцыппфшбгвхъвёълсё

Расшифрованный текст:
вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходящ
ийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпубликац
ий.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.номож
ноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднукартин
у.текстнатысячусимволовэтосколькопримернословстатистикапоказывает,чтотысячавк
лючаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблятьпредл
огами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеизменнов
озрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиилибез.учет
пробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименностолькоразмы
разделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,таккакэтоп
устоеместо.однаконекоторыефирмыибирживидятсправедливымставитьстоимостьзатысячу
```

```
символовспробелами,считаяпоследниеважнымэлементомкачественноговосприятия.согласитесь,читатьслитныйтекстбезединогопропуска,никтонебудет.нобольшинствунужнаценазатысячузнаковбезпробелов.
```

Карточка:



Блок-схема:



5. Шифр Белазо

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

Код программы:

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

key = 'ключ'

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

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

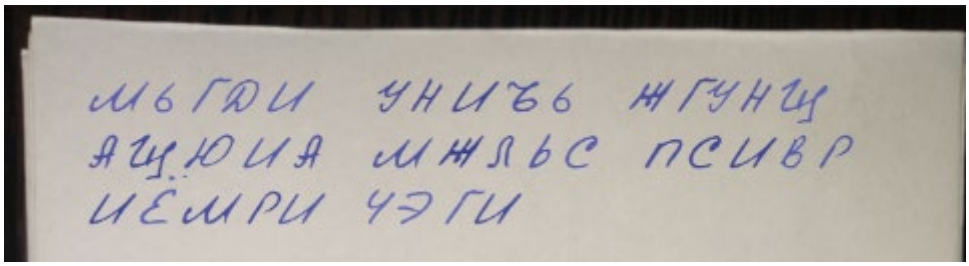
```
print(f'''
Шифр Белазо:
Ключ: {key}
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{bellaso_encode(input_for_cipher_short(), key)}

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

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

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

Тестирование:

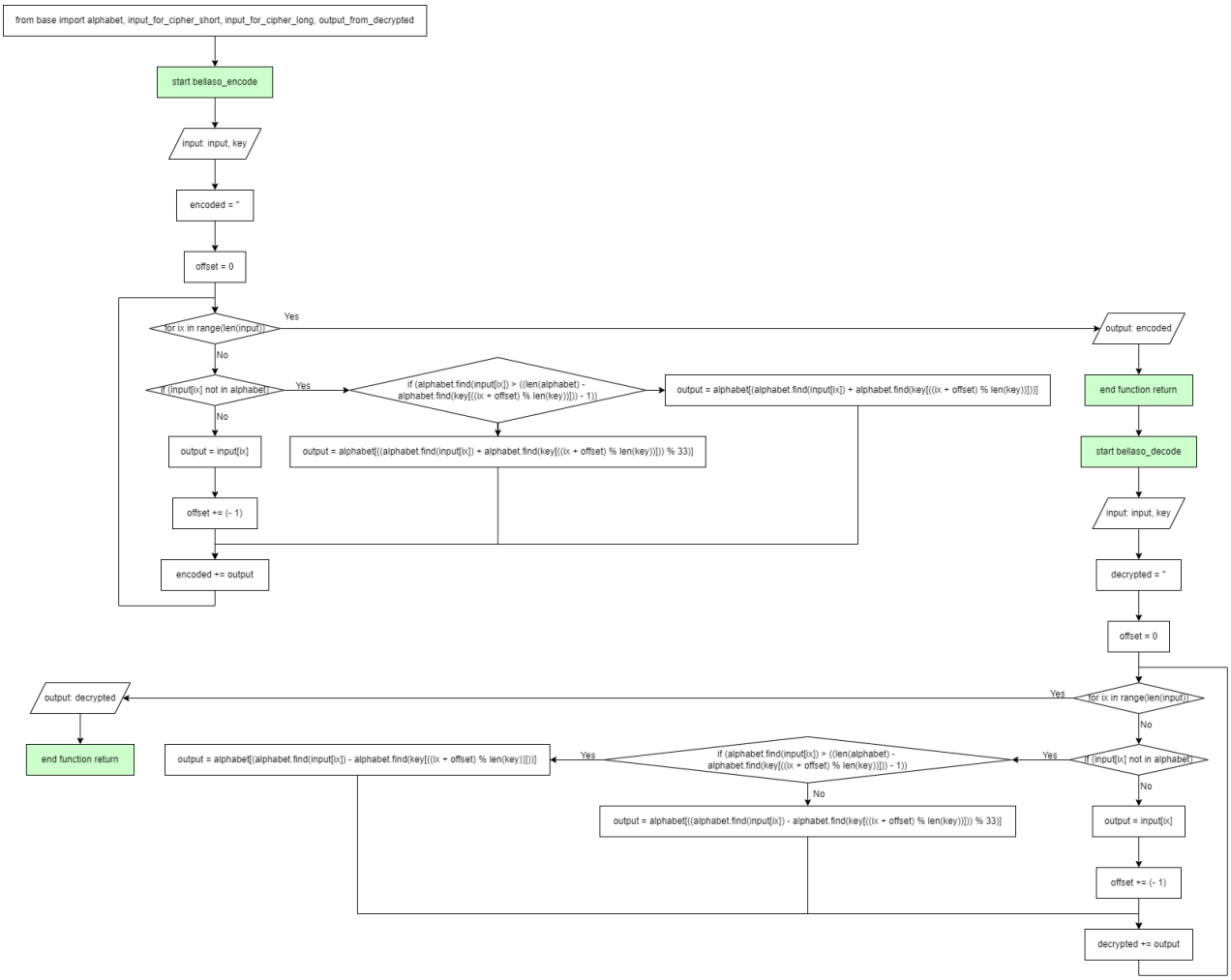
```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab02_5_bellaso.py

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

Расшифрованный текст:
время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
мържыфкьыэрчэзжекющийгсиушаёцъайвцыйщпмиэлрёвщмдкчгежцжбэриизунйщырачлйушънёоб
мыйежбочэвкьрёврийщнюзщнаашюгзшррацфкчнлёашлуацфвгйщгшщчъпубжеяъодквжёшщщмъяяг
уцюнухрохнрчхъкйпцпйпьгыхъятмлгйлъйьппакафйаэьдмкмёчбраащмщошъмыущнёоуюъщчмщщц
рохщмдщтлёумгяшрбёэгиекющийгсиушаёцъазпцмдпщвёмллёуэнёцзёёмлрущпжеучжымлигигюа
щплкхлойущсйвцрьхэрекющийгсиушаёцъафэъпвщчъвщыоачроещэйёмюхвьююйуэрахлнёхлётмл
гйтыроэъртькхчмцйхвлгймэгшйэрёъкруорпцэфйаонгиэфпгщнпзппльфнггугжеёюхвшъёжэрпг
ууйёюымйырягйюъжырвгщоюдуунйьъьякшжаоьсъушжокэрцчфоьвфлчщпжеучжымлпачнмгкунйэъ
иёцфхььюаёьчмщшржячрлещнмяылпйкррйвцавщыжзкхрьыэиёфпгцэрйушъпйуыоашкрёьгжйкюъй
ёээоуэнзщмггкшжацфяьтюхвюггйъьмшпчмщюнггугжщкррёлёгдэриизлнзушгзшълчьюмацфвщпэ
раьфкщщчмщушгешъпйщчъвщьюячжочтпггйркицъачьнмшщплтчыоёьюочшэрщщшрохэхаэлруъьмш
пчщякцюявфиашрйхлкряъюрчхцювзюмжюэрёпшгиэърохъвекцмепцмйщьщьяфодёфяаытжщупэйыы
очмрвгунщдьююущуюъиэъждщэрутлртькхкьфкщщчмщьыоёлрйччфёжээхаэлэжщэйьощжьмлееёшыг
пшгеэъквкггиэнгешъбёмъпжыфэйукрохэмъцлпаэрпутыроуюйжэйаэщщбэриизмгяппжещомжыъ
нкьцюяъюлахюмепмсыпюрохщмшщчъпущпймялксщюнпщюякющийгсяшлиёммгяъьмшпчмщэги

Расшифрованный текст:
```

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

Карточка:



Блок-схема:



Блок C: ШИФРЫ БЛОЧНОЙ ЗАМЕНЫ

8. Матричный шифр

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

Код программы:

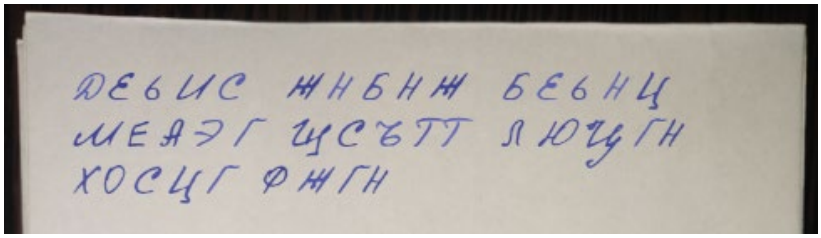
```
from base import alphabet, input_for_cipher_short, input_for_cipher_long,
output_from_decrypted
import numpy as np
from egcd import egcd

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

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

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

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

    return matrix_modulus_inv

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

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

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

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

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

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

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

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

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

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

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

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

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

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

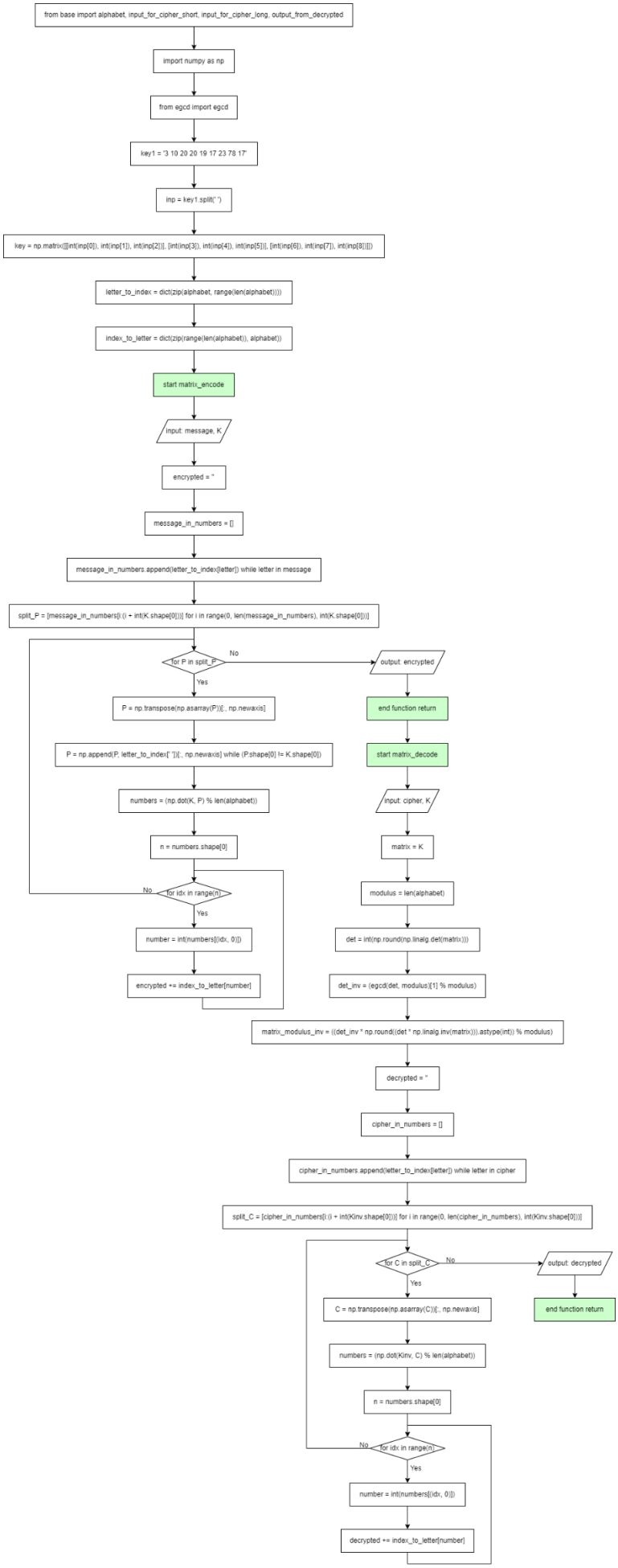
Тестирование:

```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab03_8_matrix.py
Матричный шифр:
Ключ: 3 10 20 20 19 17 23 78 17
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
дёьисжнбнжбеьнцмёаэгщсъттлюцгнхосцгфжгн

Расшифрованный текст:
время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
щвичнкфящёжщрээншусзуйдътюцбёэъяшщктыёжщжтийдмпжбзярюмсигфохжртичаужбвфэовкквк
ыкзчвзяжтиьудюяшгмювжаепыэсофрдюейхьёзпфрдчвзйсоюбёъштифчыхвлицхилщндкыбокттжж
гнщъяблпфыдчикъллвусфвпъбёъётстрэуижлиатйчлихчрозюпдмзмлёъзпжюцбёввижгнйдмфшж
тлйбншедщжгнншусзуйдътюцбёдёьбллющрюяшядмдбмйихюяшбээзмлфрдяпйшцпнфшшвкающмдуй
юьсщигфъарыгхзчэдкхтщтяшцтиоььфчлчнкфящзфэхосжгнёжщюжръчасожцповусштдбнивщусзй
дзшцпвфшьйотрцачиаъшцъйёгфрдэъпчияъжпшеьёттскиенаеэёгсюъвуефъацжщофабюязърцпяш
лнкчтшюистщоъшхгчкгёядьчтйёьчеажщъёяъьявшзмлфрдяпйэдкхтщщплгйяоььъжхлжщцэоеяшъ
бьоиыфэцъаптъхйжгнзыгвядгобомиььхеёжкрмзпчичнкыфмшэомюцмээусзлеэзнкфмедщкфрдим
ежгнщцгмфвтэяхосймфыгщмнцьвпёдцщэобзчожтвнокдызпмвщвгщзыюжрътюцбёпчръщръмяюбём
ллаьпяыогнобрчгюяшажиьгдмжзнкёчичыдгъяьиммсшмюцфммрэнохюёжюпплкяыщбяьъиохжиъож
фцшщъщьачижччачижгншвквхшёсошязцезрэлцъсшдыоемъсимзецарцётьнцфжуэфгъмммвщебжюм
мвфудпулкктиэхосйбмтэячтфтщогземюцлиьвъжщцкбвжяьбнфжччхлтмъбёючжъяётгьеобэобее
жржгнщпгхвдкнёыъчдбнмюцъммэуилэчфпыщчишюззмлбасзнкэучцпозбнжяымещкщтжвпуипоэяй
щерджбйозюьщштщрэжюусзчйдчшрозжншрэнъенэъягнх

Расшифрованный текст:
вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходящ
ийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпубликац
ий.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.номож
ноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднукартин
у.текстнатысячусимволовэтосколькопримернослов.статистикапоказывает,чтотысячавк
лючаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблятьпредл
огами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеизменнов
озрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиилибез.учет
пробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименностолькоразмы
разделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,таккакэтоп
устоеместо.однаконекоторыефирмыибиржividятсправедливымставитьстоимостьзатысячу
символовспробелами,считаяпоследниеважнымэлементомкачественноговосприятия.согла
ситесь,читатьслитныйтекстбезединогопропуска,никтонебудет.нобольшинствунужнацен
азатысячузнаковбезпробелов.
```

Карточка:



Блок-схема:


from base import alphabet, input_for_cipher_short, input_for_cipher_long, output_from_decrypted
import numpy as np
from egcd import egcd
key1 = "3 10 20 20 19 17 23 78 17"
inp = key1.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))
start matrix_encode
input: message, K
encrypted = ''
message_in_numbers = []
message_in_numbers.append(letter_to_index[letter]) while letter in message
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
No
Yes
output: encrypted
end function return
P = np.transpose(np.asarray(P))[:, np.newaxis]
P = np.append(P, letter_to_index[' '])[:, np.newaxis] while (P.shape[0] != K.shape[0])
numbers = (np.dot(K, P) % len(alphabet))
n = numbers.shape[0]
for idx in range(n)
No
Yes
number = int(numbers[(idx, 0)])
encrypted += index_to_letter[number]
start matrix_decode
input: cipher, K
matrix = K
modulus = len(alphabet)
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)
decrypted = ''
cipher_in_numbers = []
cipher_in_numbers.append(letter_to_index[letter]) while letter in cipher
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
No
Yes
output: decrypted
end function return
C = np.transpose(np.asarray(C))[:, np.newaxis]
numbers = (np.dot(Kinv, C) % len(alphabet))
n = numbers.shape[0]
for idx in range(n)
No
Yes
number = int(numbers[(idx, 0)])
decrypted += index_to_letter[number]

9. Шифр Плейфера

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

Код программы:

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

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

key = 'ключ'

def playfair_encode(clearText, key):
    text = clearText
    new_alphabet = []
    for i in range(len(key)):
        new_alphabet.append(key[i])
    for i in range(len(alphabet)):
        bool_buff = False
        for j in range(len(key)):
            if alphabet[i] == key[j]:
                bool_buff = True
                break
        if bool_buff == False:
            new_alphabet.append(alphabet[i])
    mtx_abt_j = []
    counter = 0
    for j in range(6):
        mtx_abt_i = []
        for i in range(6):
            mtx_abt_i.append(new_alphabet[counter])
            counter = counter + 1
        mtx_abt_j.append(mtx_abt_i)
    for i in range(len(text) - 1):

       if text[i] == text[i + 1]:
           if text[i] != 'я':
               text = text[:i + 1] + 'я' + text[i + 1:]
           else:
               text = text[:i + 1] + 'ю' + text[i + 1:]
    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):
```

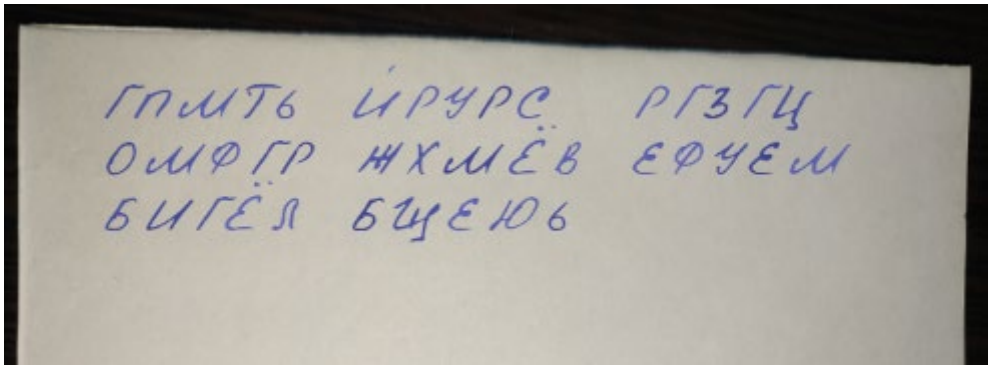
```
            mtx_abt_i.append(new_alphabet[counter])
            counter = counter + 1
        mtx_abt_j.append(mtx_abt_i)
    if len(text) % 2 == 1:
        text = text + "я"
    enc_text = ""
    for t in range(0, len(text), 2):
        flag = True
        for j_1 in range(6):
            if flag == 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:

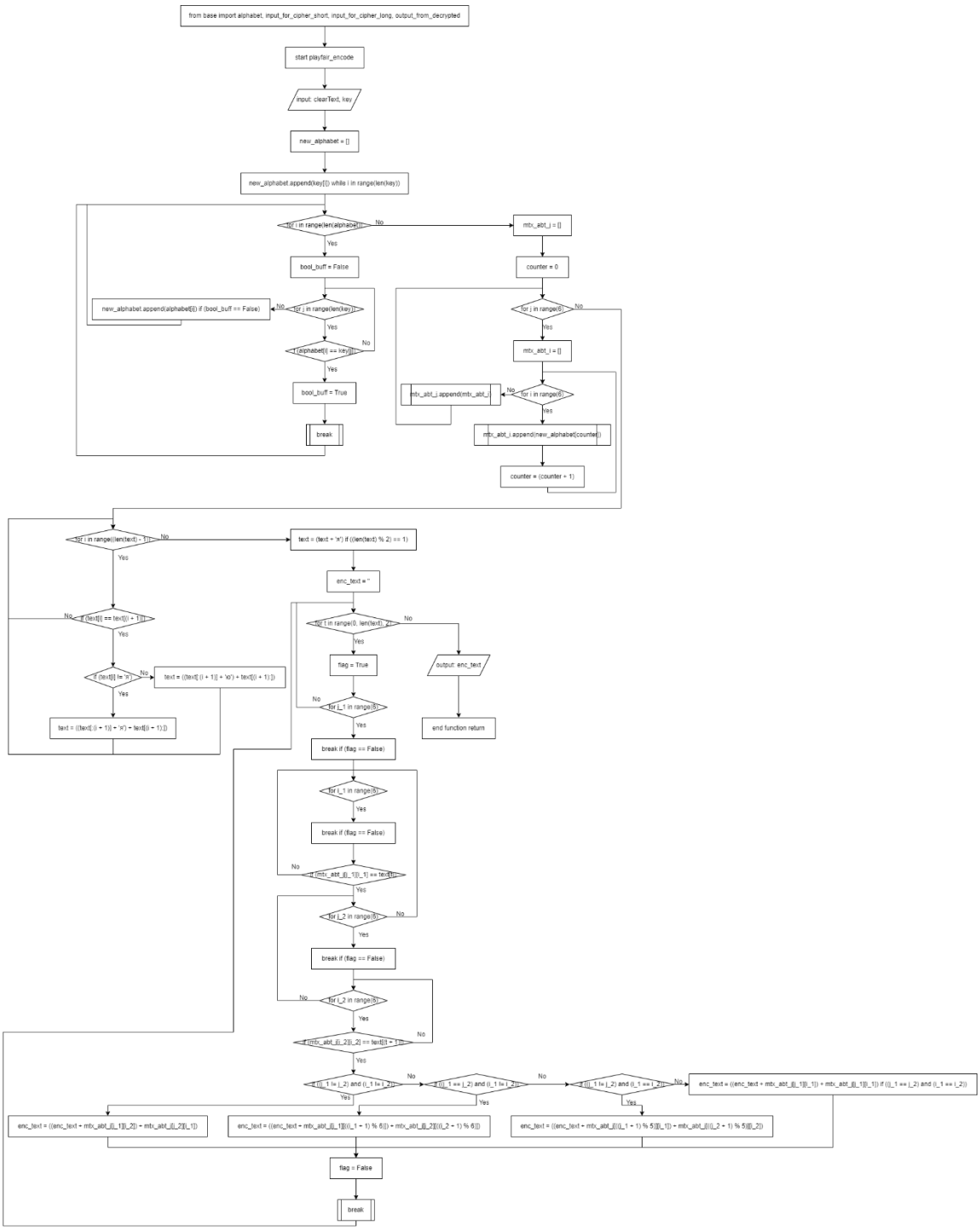
```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab03_9_playfair.py

Шифр Плейфера:
Ключ: ключ
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
гпмтьйрурсргзгцомфгржхмёвефуембигёлбщеюь

Расшифрованный текст:
время,приливыиотливынеждутчеловека.я

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
жзурцртмстучпайочушфаюфтйнжзбиепаларйжйфучфмамзнбюёмввймщмюппмрузфрмнчкэозфзвш
йжашймгюьюлуфмемчпзжлузжгзмугтмёрмгрнчёлийуёцзгргюбйжчибяхзцйоыфтилъйзннхцрфкю
зллъймщевзучбзтщвчтуггёебзжбёкмщжфчгёееёпцзгрусвъбкнкщггзфжщбозйжйофзвйлёибзжзб
щеаззнфоозолвммёжищеазчушфаюфтйнжзбигпвчзнёмжйёкозйрфзкэизёкпайжйоргйгёключалн
йжуълбсуйофуалщмюпумчушфаюфтйнжзбигьфмпюибввзфцртмуийфбиепалтучуйррмлбфзлбохёк
мщпхщефмфщшдабзвючабмщдпжчдшфмсьпаеёшдрмгрегдтрмрюзжтсёемёздгчмлйощфалозпхщмрю
йибифрмфтгкюасвхтггюижчнйируфйкйчниигсрёйнмлюусанйтглмуёйжйоргйгёкрйзеибкнруфм
бзгремтужзрюзжмёйитмннзжзиултучётталзвзфрцюнщмстбзсйдащмкэозтузрцрйъфмтюмрчуап
шфаюйррсфжгччниигрчжмпалтамщрсфжгчзжпёгчмлзгчёфмаымтщмюпучрсйнгтозуётузйгрегдт
рмрйзеибзжйнёмозтуибввифкнощулйвгчадйтбиёкпдфжйжоъзтфитуулйупезнщеюплмучпарсфж
гчхоблкнлмлзмёючюспмруучккбларзффтфммтдтфмщебзёйблзовчмфифщжротицолофгзгйгастр
улгёгюзгщотукёмряпфмйнйфпанкфщшдатрйзеибзжтрфичжюбнйпхутлмучьсйфчгёймгёкёощогг
мтёмфмзчбадтпеёмозжижзтрцрасйэщеюпижюбрйщмпяпхщемрчуяпгрумшощмюпфчвмёейоижзффи
рфпюкнруойчпзожчсёмщщеазфжибяхйотуёпуъёолъёмкнчушфаюпнуёбзжквмрсфжгчзжщеюь

Расшифрованный текст:
вотпримерстатьинатысячусимволов.этодостаточномаленввийтекст,оптимальноподходящ
ийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпубликац
ий.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.номож
ноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднукартин
у.текстнатысячусимволовэтосколввопримерноослов.статистикапоказывает,чтотысючавк
лючаетвсебюстопятьдесютилидвестисловсреднейвеличины.но,еслизлоупотреблятькпредл
огами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеизменнов
озрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиилибез.учет
пробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименностолвворазмы
разделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,таккакэтоп
устоеместо.однаконекоторыефирмыибирживидятсправедливымставитьстоимостьзатысючу
символовспробелами,считаяпоследниеважнымгементомкачественноговосприятия.согласитесь,читатьслитныйтекстбезединогопропуска,никтонебудет.нобольшинствунужнаценазатысячузнаковбезпробелов.я
```

Карточка:

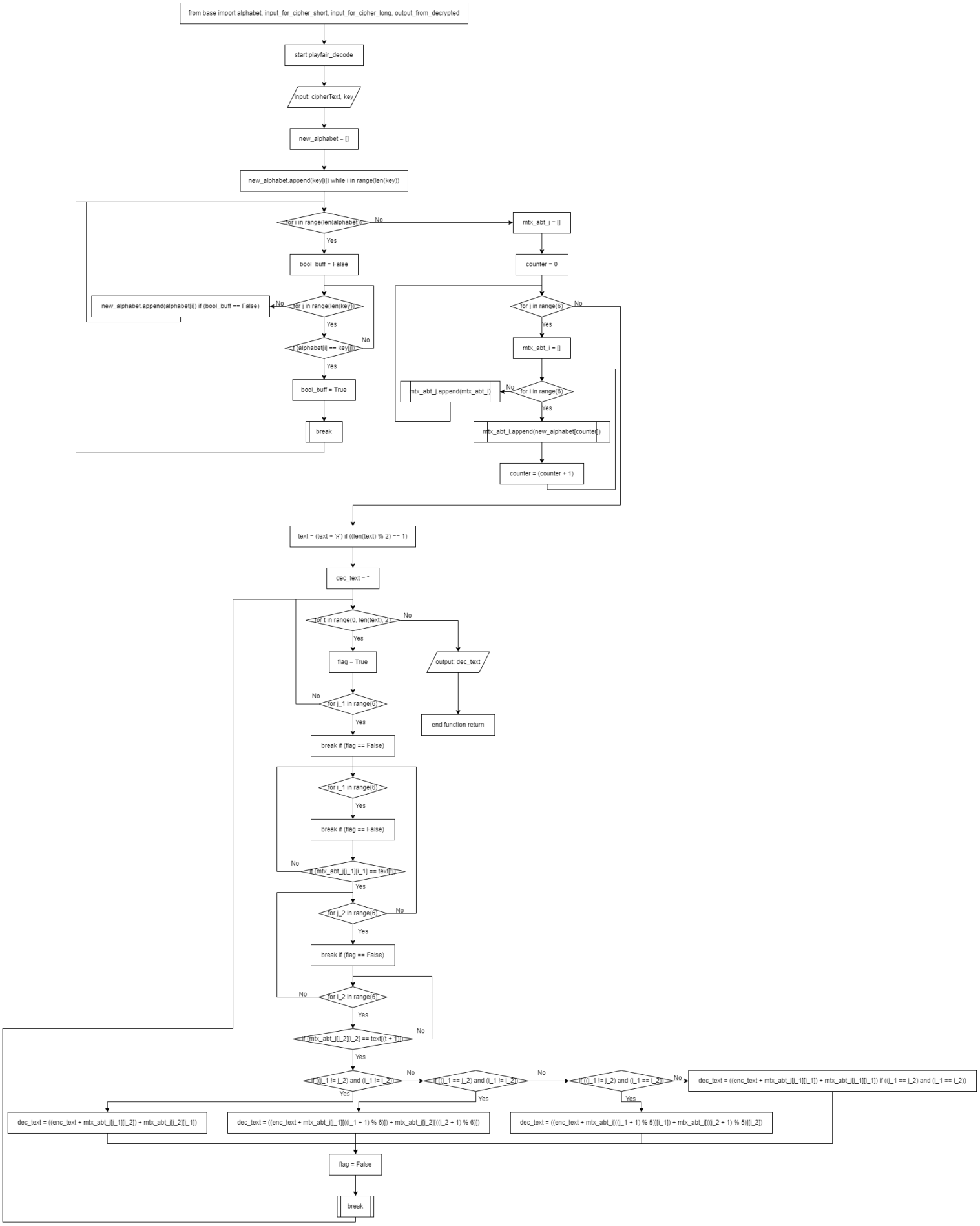


Блок-схема:

Шифрование:



Дешифрование:



D: ШИФРЫ ПЕРЕСТАНОВКИ

10.Шифр вертикальной перестановки

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

Код программы:

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

key = 'ключ'

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

    k_indx = 0

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

    col = len(key)

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

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

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

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

    return cipher

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

    k_indx = 0

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

    col = len(key)
```

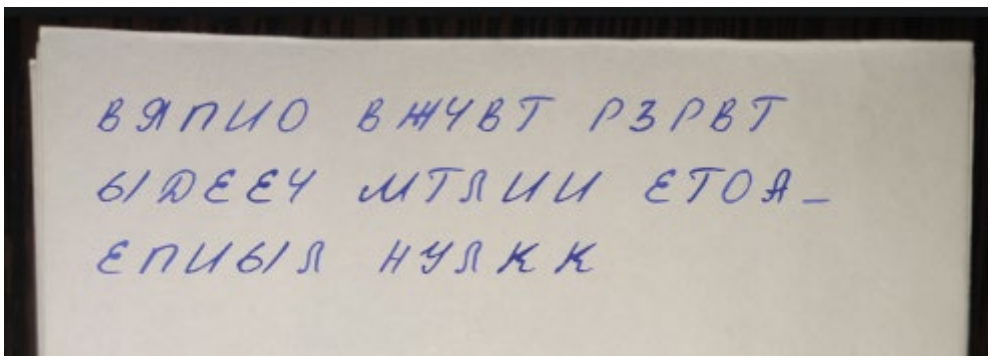
```
    row = int(math.ceil(msg_len / col))

    key_lst = sorted(list(key))

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

    for _ in range(col):
        curr_idx = key.index(key_lst[k_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:
```

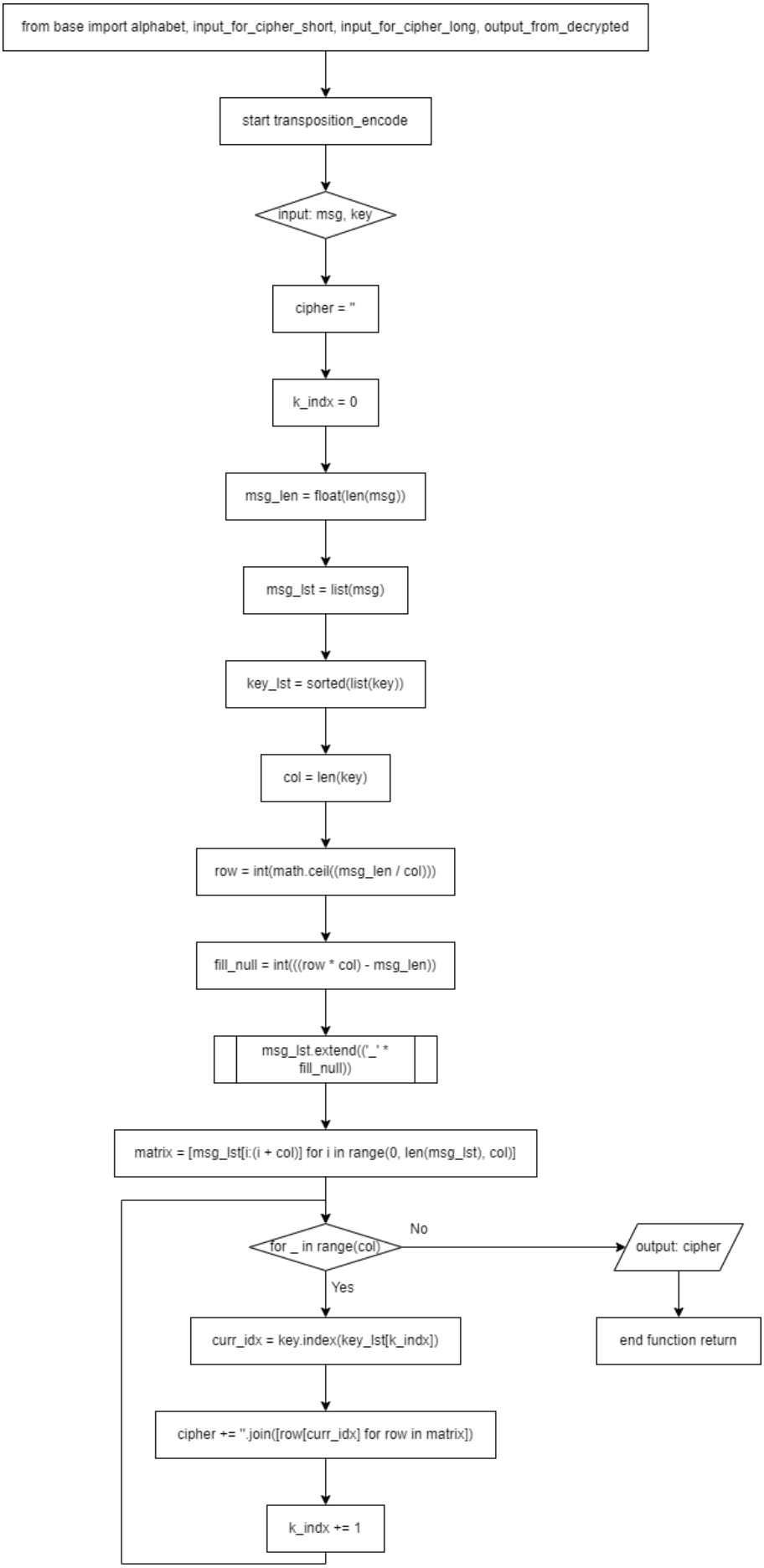
```
Зашифрованный текст:
вяпиовжчвтрзрвтыдеечмтлииетоа_епиылнулкк

Расшифрованный текст:
время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
врртаяилчотчаьттомндядачоонннлгнляоифанпиикккеекквбехтацонидоокоинтаяилеевилвоивю
окичкаяилтоомовсикквзтсвчвяпдтдтоейиынтиуряроисадиамчоивматлсснморачоарйтнинса
ыиоалзупеуиабттинслесоиноомзялсомснокпеачнбпкэуеткаеофырисвиситозссосбмттодвы
етатнвриклтзиьтттеорспкеекоивжеаянвпетоисьтчмокданлкезпаохщлреввтеиааинлхоцнук
йвокроаодирбебонзлкнжбечтчмокнасьадлачданкстчмоолпесттсаааповякассяеививдвчтоез
петегзомрмсиидлавзоитлееваекпйсдеопячтссбмитчрлвчеъеамотисилмолрыдеовдптсмсарл
киеятктсмооккрииждневттосаяилпеисаснаммочвооиясаептснебдгокттбтнлнуннтчабрлчпе
анссоттсомнйстиьодйкотририаиилбшнмохлцчаттдытеуихаичдогвчмозонссормоооьндлиут
тенссоэккиноктиоытчыаютбоьяислрелнкплотлплмтзигчяеандилтоеовзнзтткреояьтиоттчр
лиектбвлвомсрраививвнткзалсабыоавчиьбззиюзакпосчнntемивталмвсмьыуввоапипеенлнк
сноптчгиьчтийсзнпузиндчбштуцзсзозбв_тмтиыуввэотоеикптлпоияткаветмзхдеьириыбатт
мсебелвлёзвыопаотонегкыувводнпзтииканруттыуввсьррлчатпзеттчлееттслесснеичзслоб
ьдапюиуитрниисопкчвоиностваткелсртиьяпеибчеооеитекпенодтмоесьаремвонрртчтоыак
лттаотетдооырбиярдыаьиттчморлзчялижэемеегсятоссталыкеиопаноутоьснааыукеоок

Расшифрованный текст:
вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходящ
ийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпубликац
ий.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.номож
ноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднукартин
у.текстнатысячусимволовэтосколькопримернослов.статистикапоказывает,чтотысячавк
лючаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблятьпредл
огами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеизменнов
озрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиилибез.учет
пробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименностолькоразмы
разделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,таккакэтоп
устоеместо.однаконекоторыефирмыибиржividятсправедливымставитьстоимостьзатысячу
символовспробелами,считаяпоследниеважнымэлементомкачественноговосприятия.согла
ситесь,читатьслитныйтекстбезединогопропуска,никтонебудет.небольшинствунужнацен
азатысячузнаковбезпробелов.
```

Карточка:

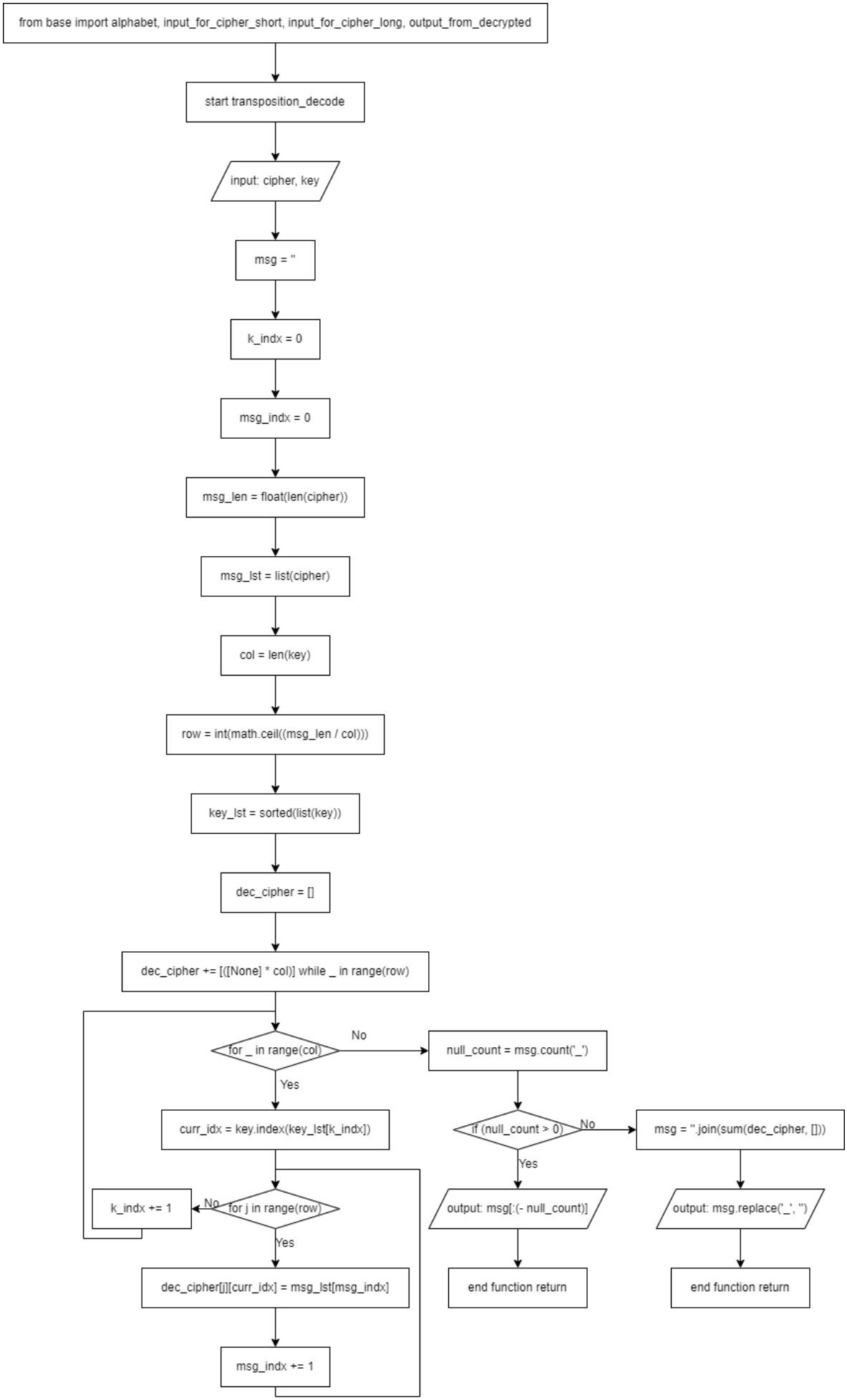


Блок-схема:

Шифрование:



Дешифрование:



11. Решетка Кардано

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

Код программы:

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

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

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

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

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

        return uncipher_text

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

texto = input_for_cipher_short()
```

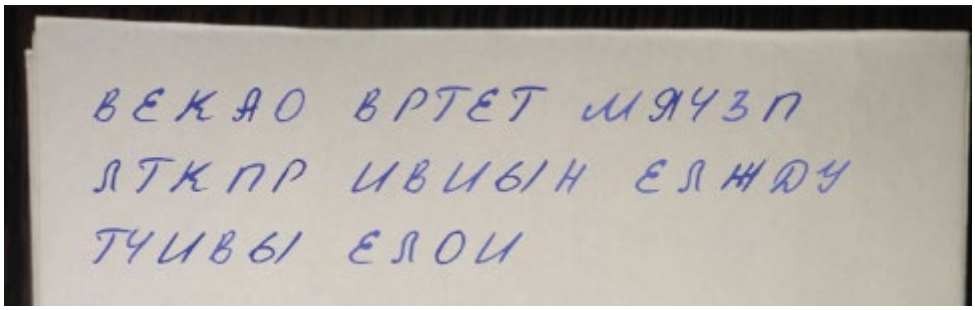
```
n = len(texto)
encoded = r.code(texto)
decoded = r.decode(encoded, n)

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

texto_long = input_for_cipher_long()

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

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

Расшифрованный текст:
{output_from_decrypted(decoded)}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{encoded_long}

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

Тестирование:

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

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

Расшифрованный текст:
время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
чалекэтанвоькийотттперкдсимыетосрсстзатятчоучпнсаимвоомлттьиовтнолимваряаопгаз
ионтщиамхвиалиьлвинойнитедплряндекоартотичлдхоектдиерейтчадянкобыквецбаоевтльи
шбтаихокоомнинтыелкхнпублстиеформацмдожнзагцоедибезонвеуехглоилвитовтриочктоёб
чыккнчхноодоминабзнпоалвакьзоелатючаивоыксдянаучуосктьиммоадиевнндирлооваиоиди
тловспоровтчлькякинстатоиучтсчтпиктуекрикссмаеритмнвопсонловэлотоатыосксядвест
```

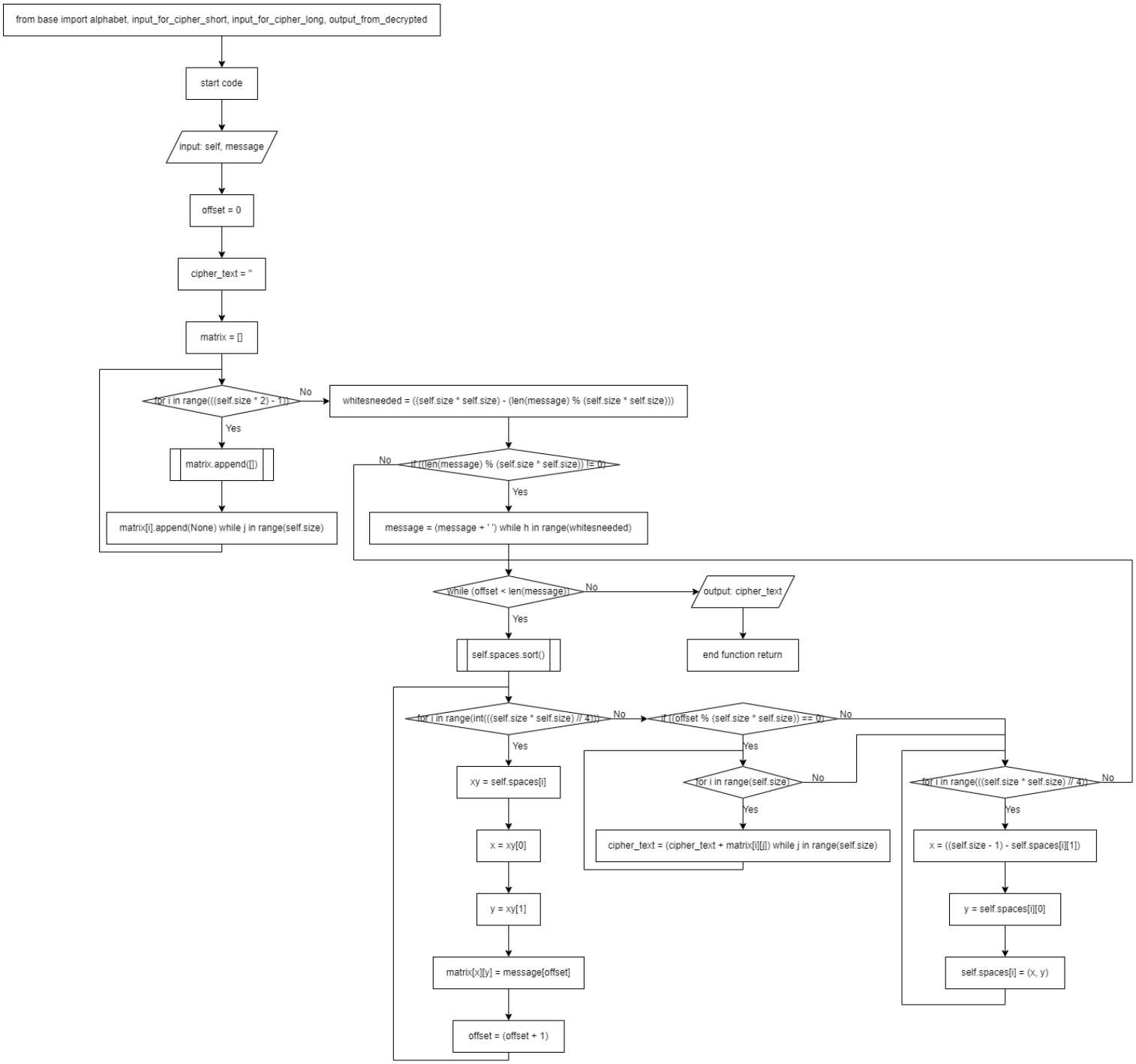
```
осскатислпозяыввсяраечтетьзпаддесвткялтниюччаетливетотсебыясоютьптзйвамиирдеел
риуегчиснидлытломгаичзмлиизокупотптрчнозеблптнеиокоизасменнлотлявмоизириерчечи
дсатввнаоссслиамволоватодизптниысяприрчаеиспрнотстбчеялквккатоопосмчийидтеаитя
ртелььтниайтостебнонъемеалистоитлблеиздевтчокексучвтсапуевреитмлтичивераипроет
обямпремсоосистралнмлвсотовлоьвоваимксмвооербаотдзнмыранызчносделтпеметтааскст
отчккчкиотдкнатаьаакпрзэкточоипкуосибнелютобнелыятззиоствымрьекзатыссожтяочтур
ыиесавфивиитьирдсятмотмспраимввыибедливачеажнтсолтвеныносвосгмопрчовэлбеиеомет
ланятсопаослемкдпмизниептогоныйтприропутсяеткиаезятсчпкскксьттбезопзтендчгитати
ньиласслииевбеназозктпробаеолнлеотвбуьдтысетшяччуитнзснкатчвунукож кнонацб

Расшифрованный текст:
вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходящ
ийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпубликац
ий.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.номож
ноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднукартин
у.текстнатысячусимволовэтосколькопримернослов.статистикапоказывает,чтотысячавк
лючаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблятьпредл
огами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеизменнов
озрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиилибез.учет
пробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименностолькоразмы
разделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,таккакэтоп
устоеместо.однаконекоторыефирмыибирживидятсправедливымставитьстоимостьзатысячу
символовспробелами,считаяпоследниеважнымэлементомкачественноговосприятия.согла
ситесь,читатьслитныйтекстбезединогопропуска,никтонебудет.нобольшинствунужнацен
азатысячузнаковбезпробелов.
```

Карточка:

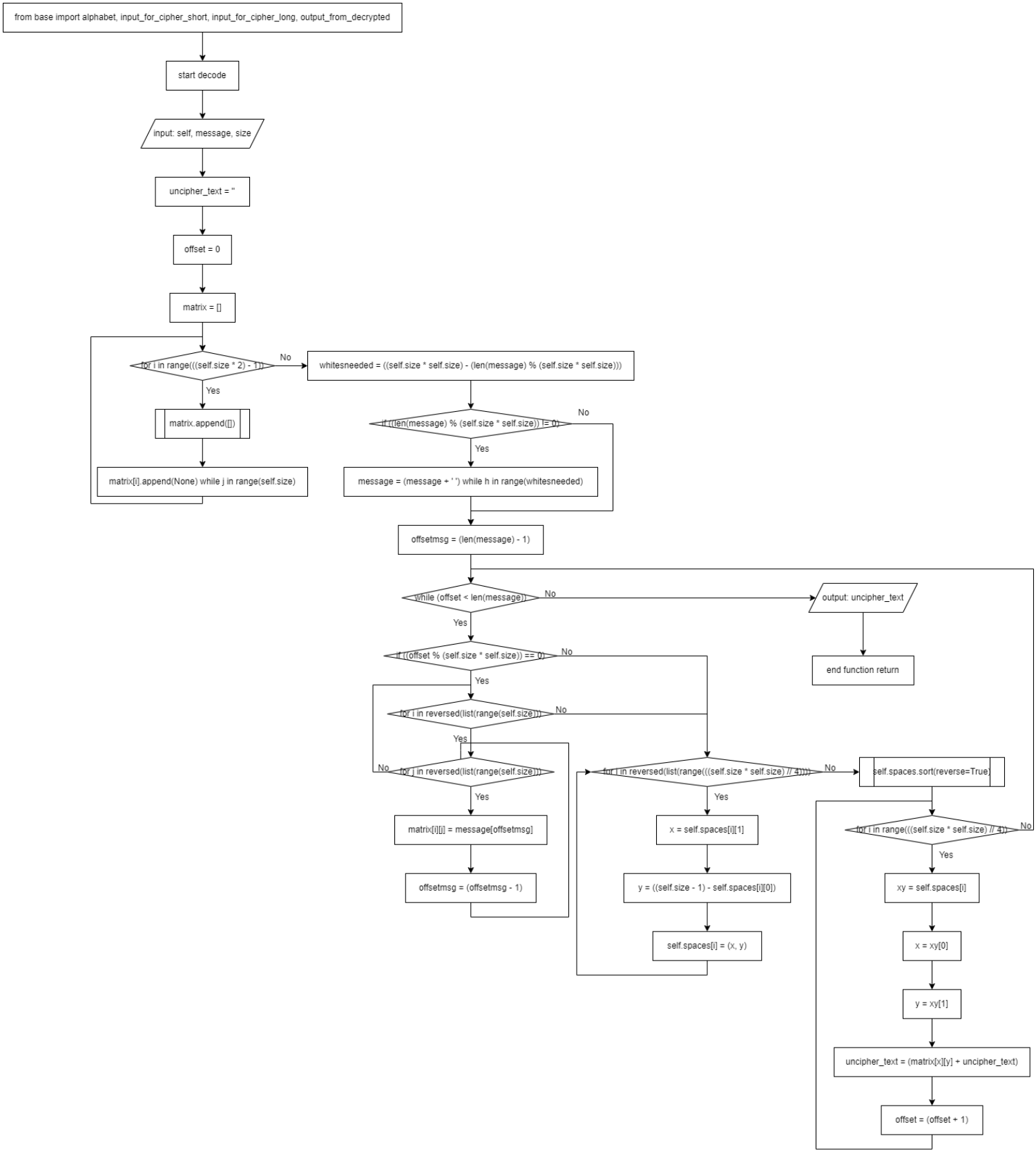


Блок-схема:

Шифрование:



Дешифрование:



Е: ШИФРЫ ГАММИРОВАНИЯ

13.Одноразовый блокнот К.Шеннона

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

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

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

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

Код программы:

```
import random
from base import alphabet, input_for_cipher_short, input_for_cipher_long, outp
ut_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:

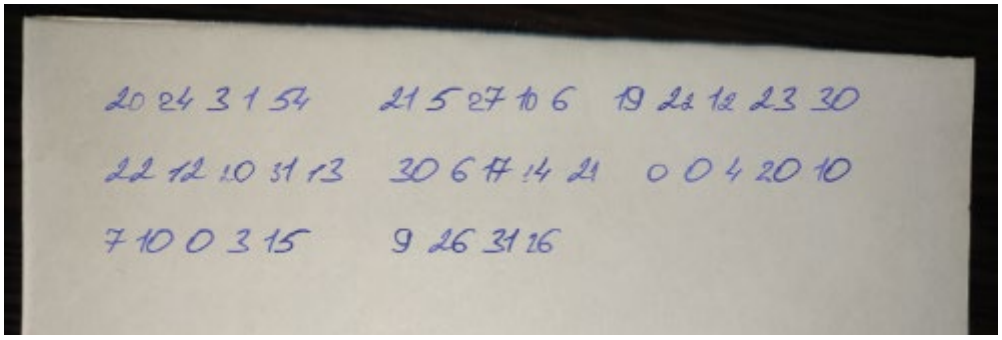
[14, 32, 13, 13, 16, 21, 1, 8, 0, 14, 29, 8, 27, 16, 30, 23, 7, 29, 5, 21, 17, 32, 15, 25, 4, 29, 8, 23, 5, 1, 1, 9, 14, 13, 23, 21, 8, 4, 9, 23, 20, 31, 20, 3, 20, 5, 14, 2, 32, 21, 29, 10, 2, 26, 14, 3, 17, 13, 28, 27, 2, 25, 32, 23, 12, 3, 18, 3, 12, 4, 23, 20, 23, 24, 31, 30, 25, 26, 2, 7, 15, 15, 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,оптимальноподходящийдлякарточектов
ароввинтернетилимагазинахилидлянебольшихинформационныхпубликаций.втакомтекстередкобываетболеед
вухилитрёхабзацевиобычноодинподзаголовок.номожноибезнего.натысячусимволоврекомендованоиспользо
ватьодинилидваключаиоднукартину.текстнатысячусимволовэтосколькопримернослов.статистикапоказыва
ет,чтотысячавключаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблятьпредлог

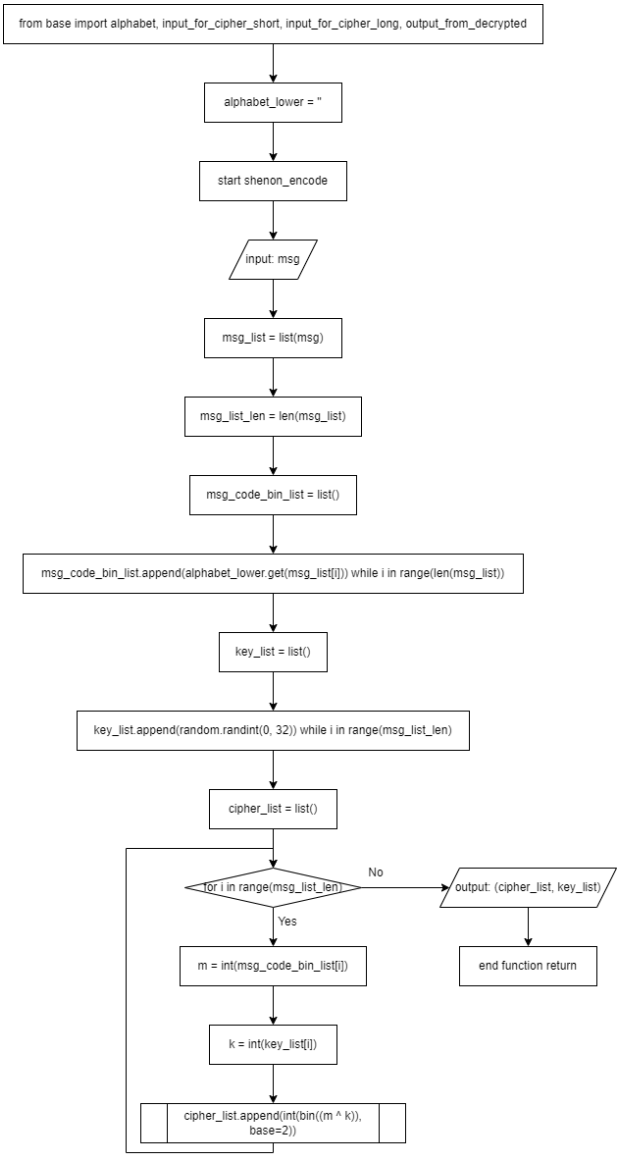
ами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеизменновозрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиилибез.учетпробеловувеличиваетобъемтекстаприм ернонастоилидвестисимволовименностолькоразмыразделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,таккакэтопустоеместо.однаконекоторыефирмыибиржividятсправедливымставитьстоимостьзатысячусимволовспробелами,считаяпоследниеважнымэлементомкачественноговосприятия.согласитесь,читатьслитныйтекстбезединогопропуска,никтонебудет.нобольшинствунужнаценазатысячузнаковбезпробелов.

Карточка:



Блок-схема:

Шифрование:



Дешифрование:

14.Гаммирование ГОСТ 28147-89

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

Код программы:

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


class GostCrypt(object):
    def __init__(self, key, 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 =
183182793879123879127893789123798218793879782387932788723783298329823980230 31

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:
183182793879123879127893789123798218793879782387932788723783298329823980230 31
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
567540261451836962860566905831140964633059962168239725400849570714503615166865
346335420862814981341444458694338340017794062562381816998355678753967115290745
875600125177595186371409630906 82

Расшифрованный текст:
время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
266050417938476356087017230014873643206160447996114515129091774959747033405943
890343897142869704450815393683605023684873888004909829680452295028718546978021
649941177826880151732274533643864532320796630272450746837347684283063682350895
557119542257601405589120123495191711757441077720644314230550270347750883998451
898671458314340016425755013995595170529190526024934975357719363564132761031905
290355995297001142437984799362147596854803459571501765077526110498459950037581
```

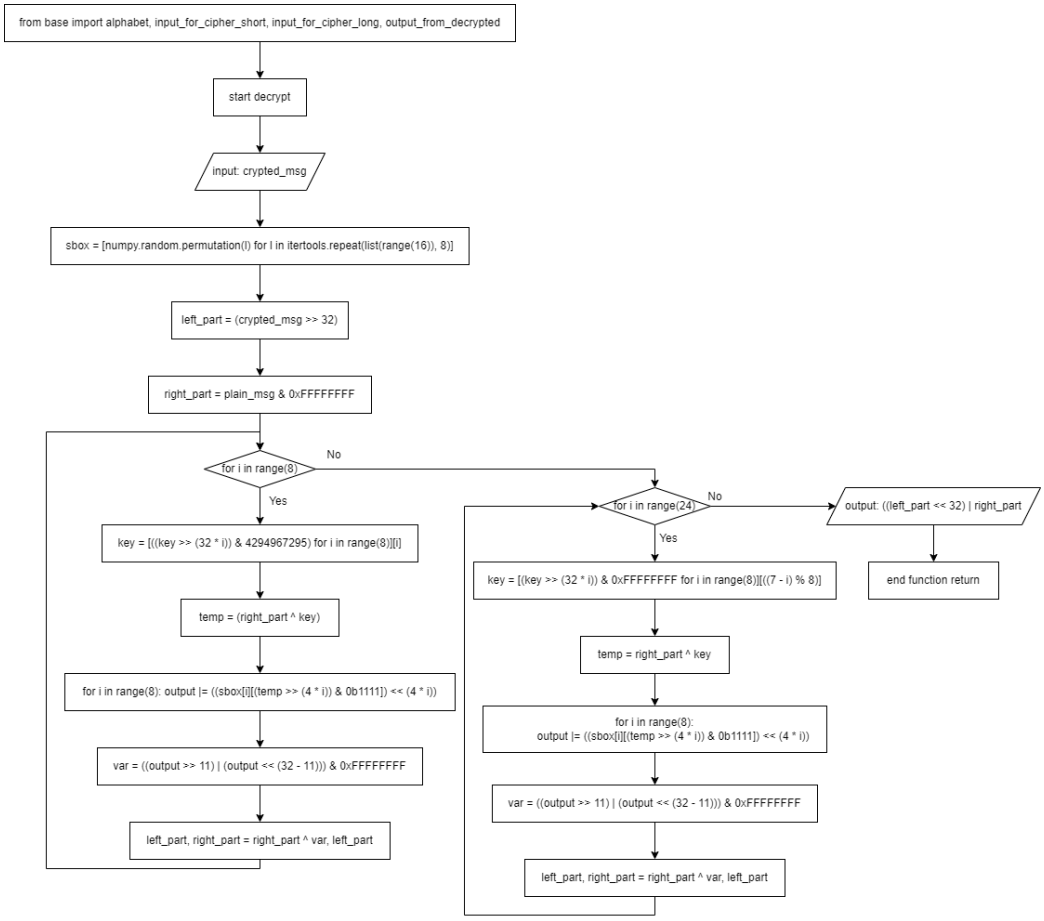
4441113013017875969917042030848511398372774550996207392191339181039778867511513245777658070846405184637307157764373019723472702707385419563060306821034731763551154503824714264812313316130706264074393244483028493575194280183695841632846694677441664704015987166764373865851781422124591521492824680316201781243695172538536956831291069312772130538659077026628200178989017827102455811284644197446147728227370823795695828448846841904902145096572681358073870106763470339835923246436026070889038200287505359250682058240850262033170450161709526251248889202783539456604116848059039836482794508570635524117505841700390724701975258248279406107572222767582208189240451665976152382603851822752929335645908960132784104671236312325136002226758406236192110306294400818405951215215170099869890409677082940113502073956343581542790367957821887506781091409598967321936432916690830836157099844286928789002776683458441918340765274704251161726938975327301299827924533461687715964034781601529646429568161764717204759432825169398902206756041104866390413896194971752446779789219834868520961790344530438359259087681089419095818947367924242461052662246177551436759868060964413164161379298518749743822796623991927620430286913534792524472501385616062283433788266136034182441382596852828678815464248898242655956549987693016474857317967045765805402303929238917825768453421398261020477598060009464910092710806727389232375505337101980972052186720533409389553482212142338388825165064316873200362417005979383125608949135305078202880494402590429918423534867498481325002725186208672068804910219444128725412313172857315960092225568479643477408050023889229147527678395405300014514788559927920823504740092277359994838446144638292221204810273531732178907255974872373943777174489278720360030245513206396286680923132665591932957479381688881713391447550197512421414080155618078837436716254239495738949352604395115116413190410600237037255778849942318131400487867993716612245815508666788386611535983488700840787806608061738649595754409762598103992609851236390147034058997120081926015140106802936589820873919900023139596870752250532835059504769730299189404268942453718027773910730364364478309783557301001910299876928643382267779773297344160509407488050435104053968601803987961053508330097865866726566439196742557135164782260908456730128332911775983685320723472682806496460888618031721287732914731095065151066593148360400366560924904292376661781984325986153706049185968639527031555339328097086138493583066405661508631851879253262157549893336342085850379421169485707846412390764082738059775423863152365222207358370420848642329032896143477728147462873088301241858500518352670460459172294780626495853313915274281775969597939390408588921013191052753639133045535705372177150216764101504465020904745448395363206321778018466643874865465360874268566642305436776611825466962196427195722777111044656000900826008555466570413212540474867009345483675013948460818856547409223615702492801811515171154353076248616600260838281861827510318136996678149054655293501452211334327394481409806145693576184764835508628555563178166027833341769254061164490913955537268308991417425837357068599913797622693757837464689346947323945655806445644037595726565639664081606635982477660096856607230205524199717614008226865423733917323140685928071900336012766976449108365776366982263306482365537014818994484990760550044356360814906973014144700157278635304496507193848832081739398232735091487075714916851955729824486796972318416181286651420560936054534723880151398796895076146362679321996134948186011756944951068239657356798522484533403233228847373198474221636636833297277075078652181416395799131787214687970115636750588484044324203374364928440353430873002856557844889902846874936816595913309923492206525652964107796613497401751530613052886454175587861865333282828775568559482244177712873569788694664109124311168154523811655185824430642639283684610406473107684254357881681853047915215675912029044604255986897798394652898650750455488036258790990498804577566340962162875175833798508271140942812797251869597190477167989914639902016960015953949566610304464307884096724577410585977044204441237724476549339641978774215508633537350

```
263474448018184154889873746151758484429412090320498815046405341806128814394905
868219934450400886138337839240264255614068324070696593204945639821400860155559
484459548932550840102882387267858402968600171176157376831434777201707285299833
699125037728269831720745214081011576306227271226229939415986451030780497468026
208293099379288210543447284509744576781872462063625033331957011656270150086434
012363118225202497856559102049032543407053333746197469855990322303022144902536
648886278160402357782913621359570037109561666619129528015842754809484320410643
238137797569327685399314897468891322774686848219935886576012631365200360985488
2178324694

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

Блок-схема:

Шифрование:

Дешифрование:



F: ПОТОЧНЫЕ ШИФРЫ

15.A5 /1

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

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

Код программы:

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

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

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

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

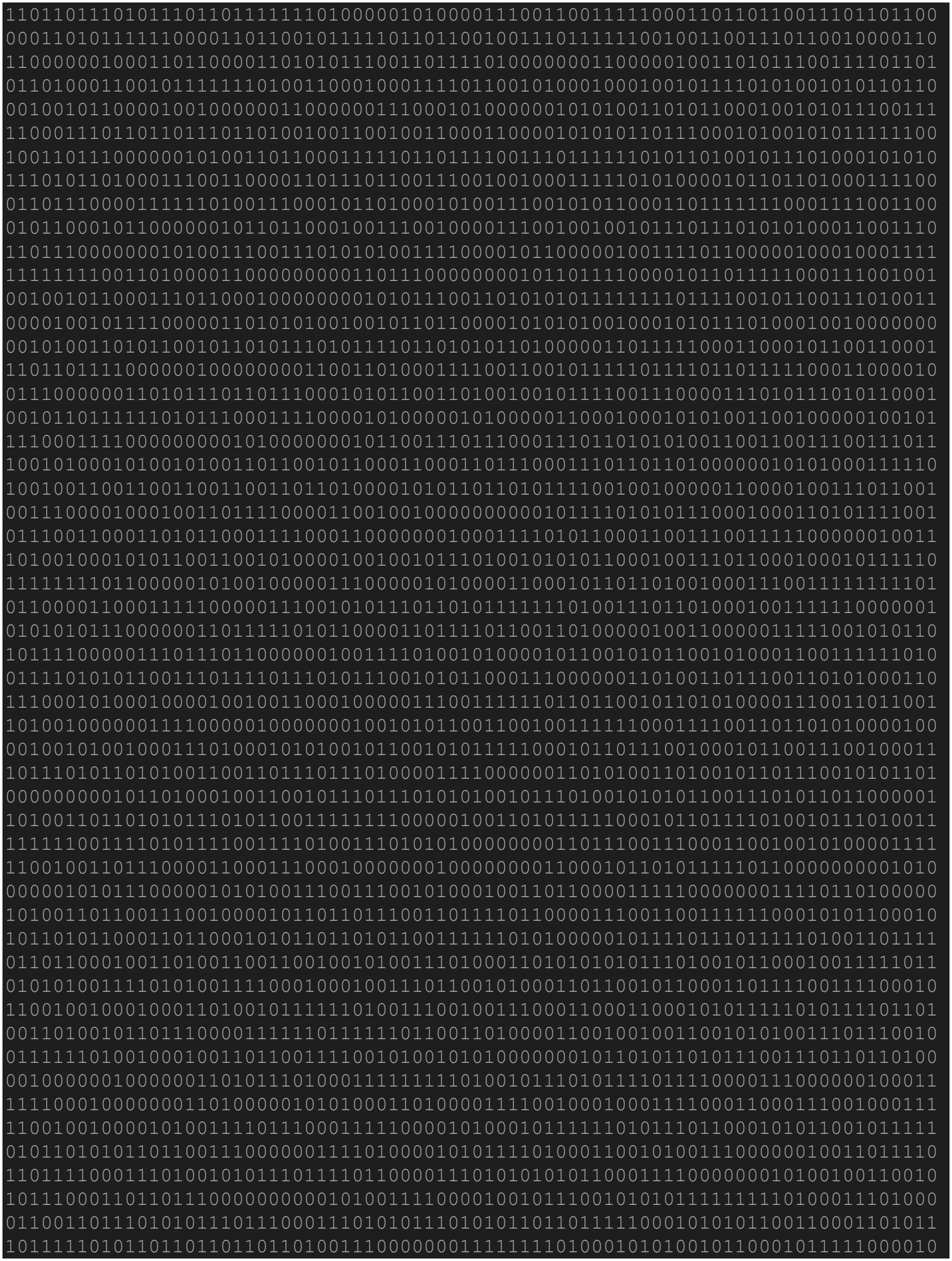
```
        key_one = key
        loading_registers(key)
        return True
    return False

def get_key():
    return key_one

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

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

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

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

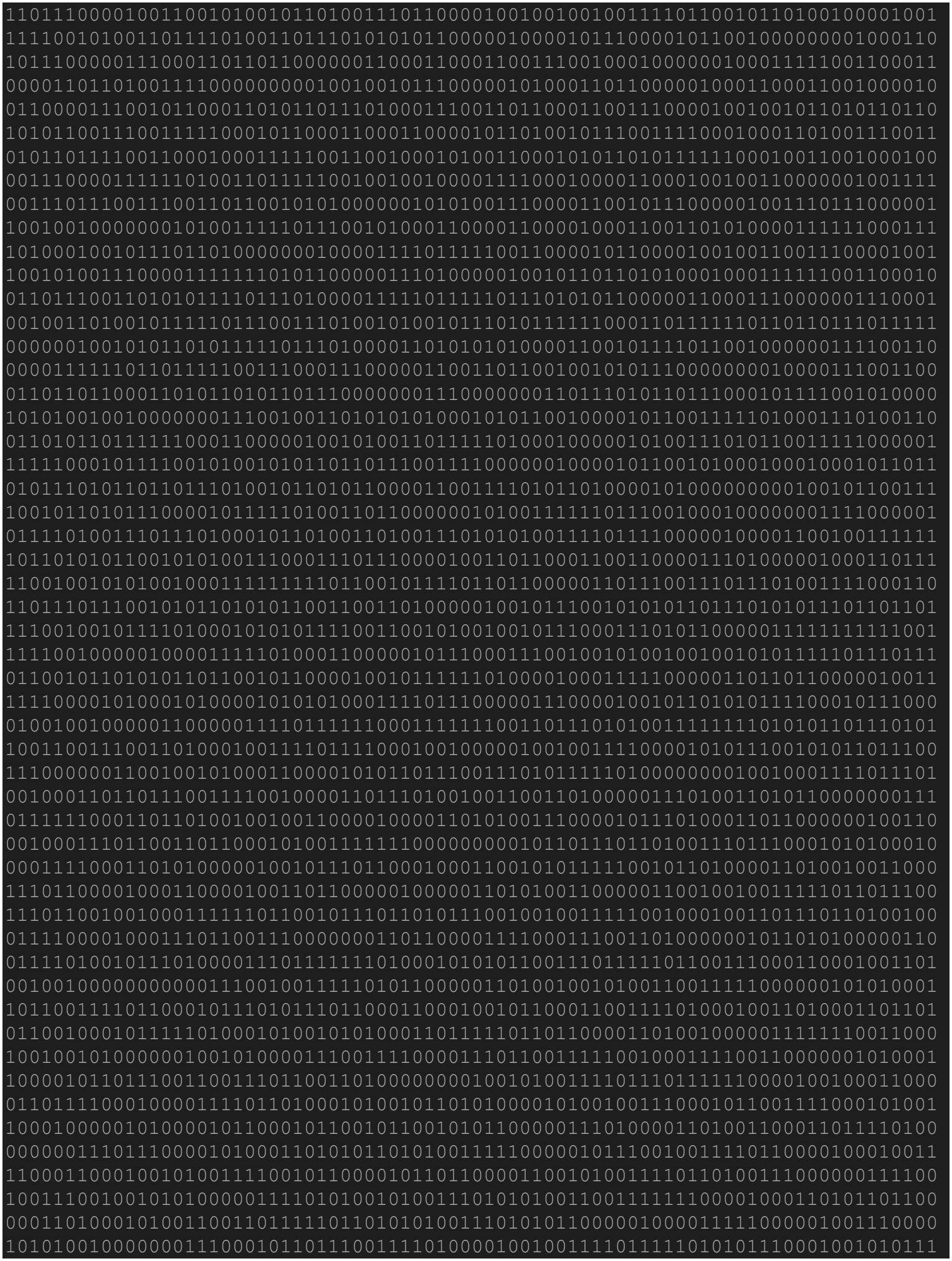
```
            while(k < len(reg_y_temp)):
                reg_y_temp[k] = reg_y_temp_two[k-1]
                k = k + 1
            reg_y_temp[0] = new_one

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

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

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

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

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

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

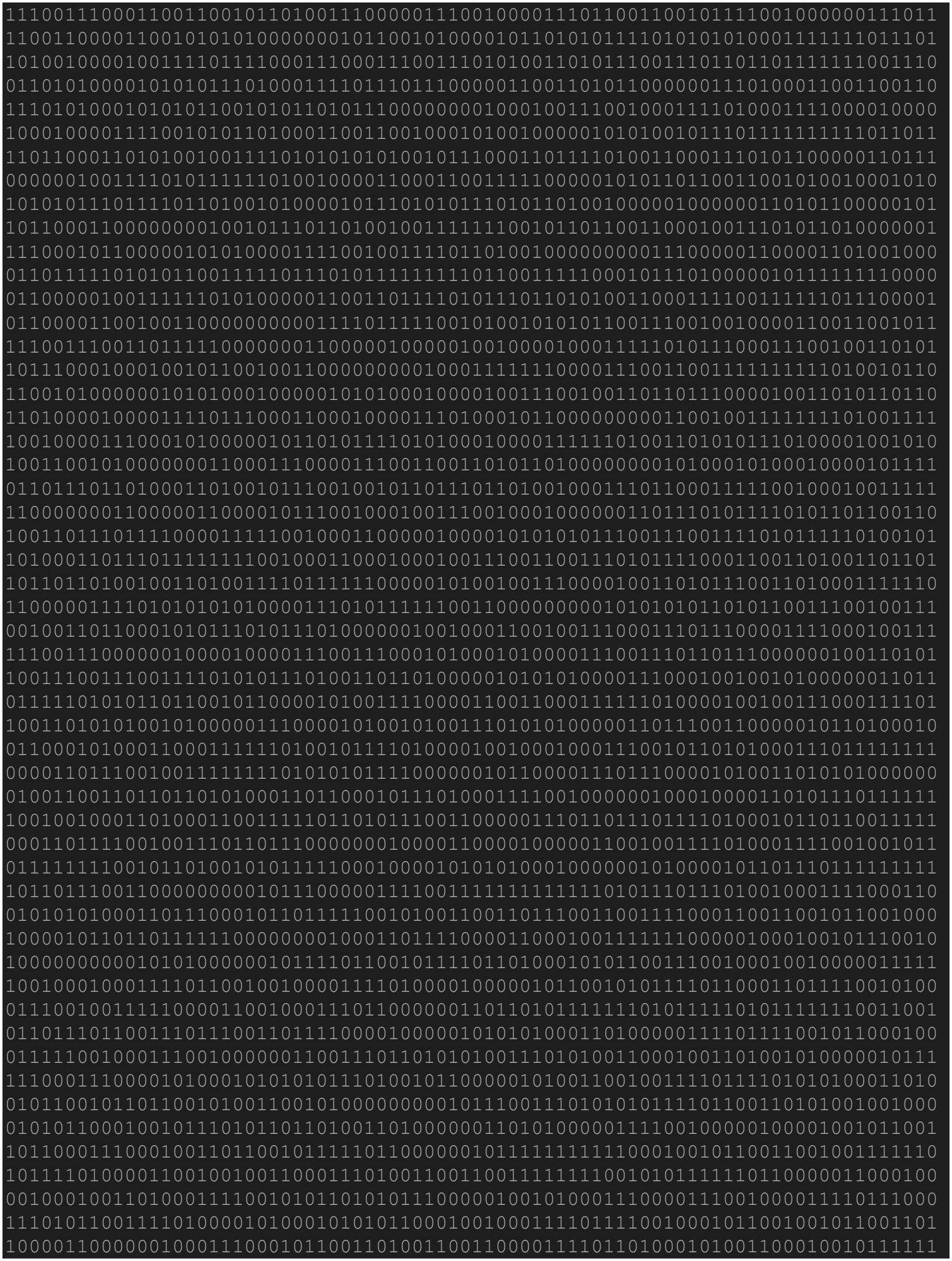
```
    else:
        while(len(tha_key) != 64 and not re.match("^([01])+", tha_key)):
            if (len(tha_key) == 64 and re.match("^([01])+", tha_key)):
                return tha_key
            tha_key = str(input('Введите 64-bit ключ: '))
    return tha_key

key = '0101001000011010110001110001100100101001000000110111111010110111'
set_key(key)

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

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

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

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

Тестирование:

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

A5/1:
Ключ: 0101001000011010110001110001100100101001000000110111111010110111
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
110110111010111010100001110101110010000011100101011111001001110110010010101100
000101101110000111110111101000110110110011011100000100010011000101110011111100
110001110110110111110101101011100011111101001110011000001001101010010111101100
100011001101010010111010010100100010001000010101100101001011111011001010111110
001000101001010011111101100001100000010100001001101001100010000101010111100110
000100110110111000011011100100001000010000110101110111011001111010111111000
Расшифрованный текст:
время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
```







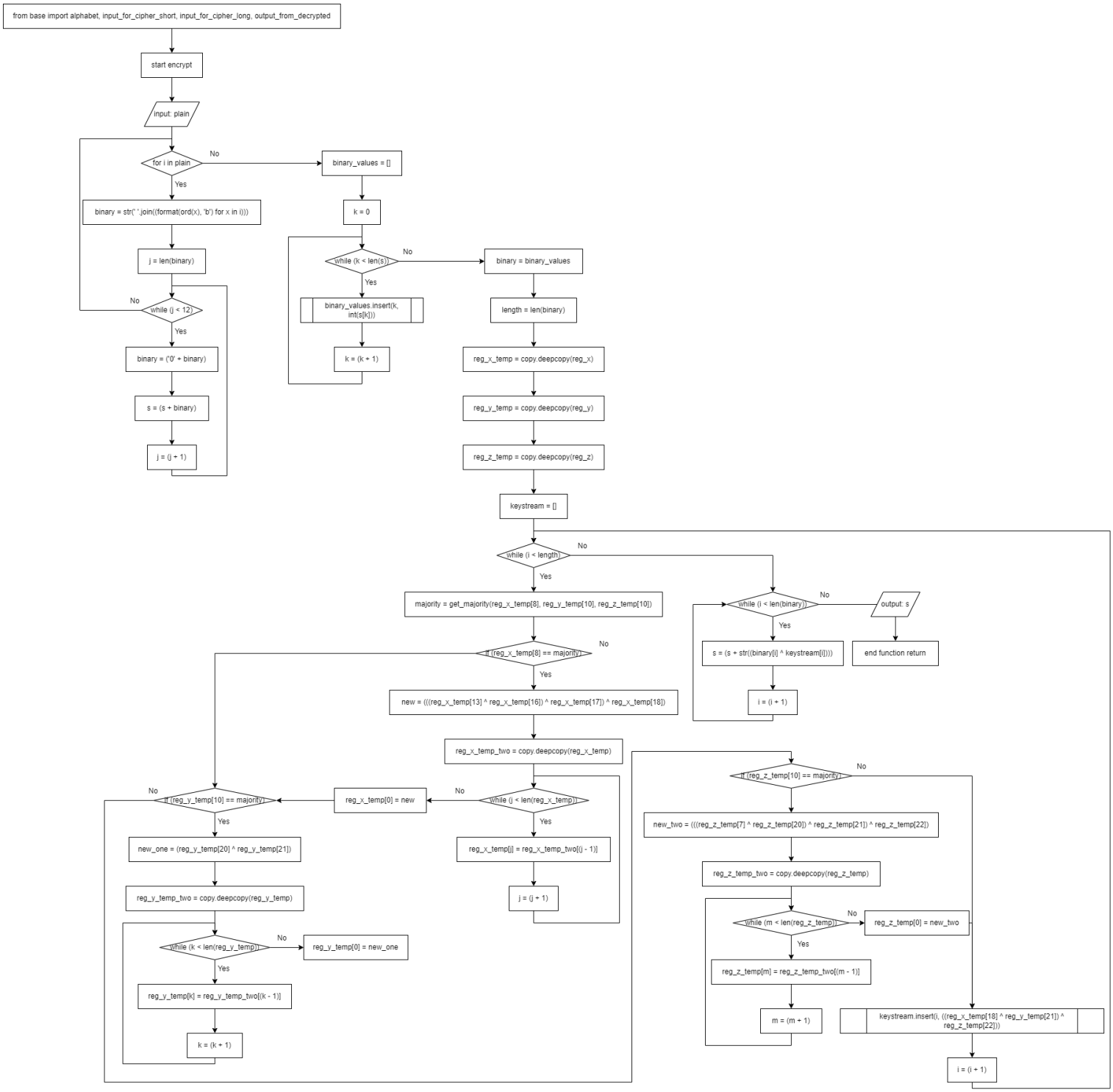
```
1000100101000001101101001111000110001110010111010100101011100010101100111111111
1010001101110000111010100101100011011100111101110000010010010011010100111111010
0010001111000100000101101111101110010001010000011000111101001010101001110010010
0001011110001001011000110010111011110011101001010100000010000001000010101101011
0111010010110000001111110101110001001011100011101100001010110010000101011001100
0000011011110011001111100000100110011111110011011100100011011110011011010110010
0111010100011011001110001011110101000001110010001001011011110001100010011011010
0001001100010000011010011101110101110110101110111010010011001100001100100111111
101010101011011010
```

Расшифрованный текст:

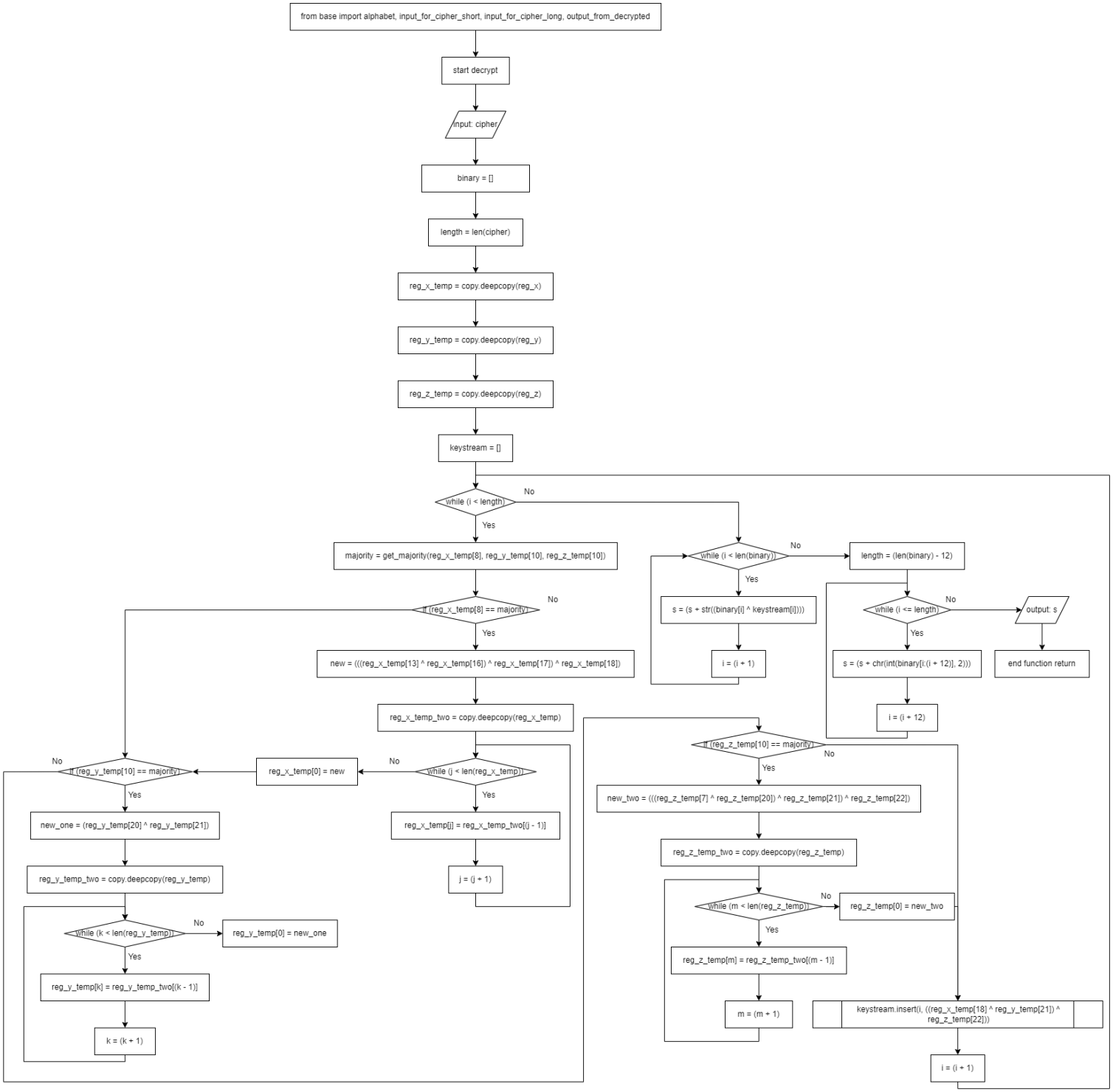
вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходящийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпубликаций.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.номожноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднукартину.текстнатысячусимволовэтосколькопримернослов.статис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

Блок-схема:

Шифрование:



Дешифрование:



16.A5 /2

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

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

Код программы:

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

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

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

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

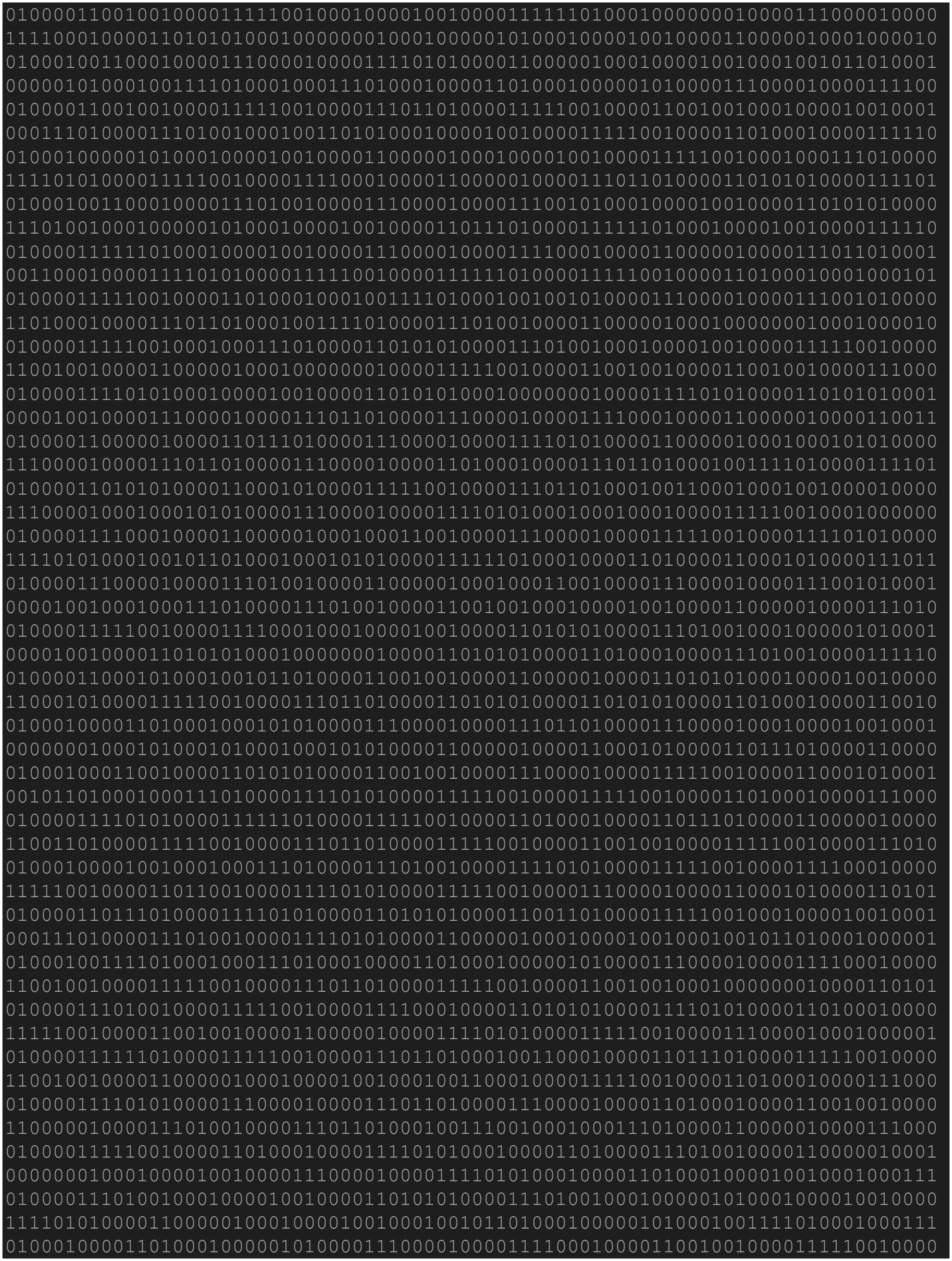
```
    while(i < reg_e_length):
        reg_e.insert(i, int(key[i]))
        i = i + 1

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

def get_key():
    return key_one

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

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

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

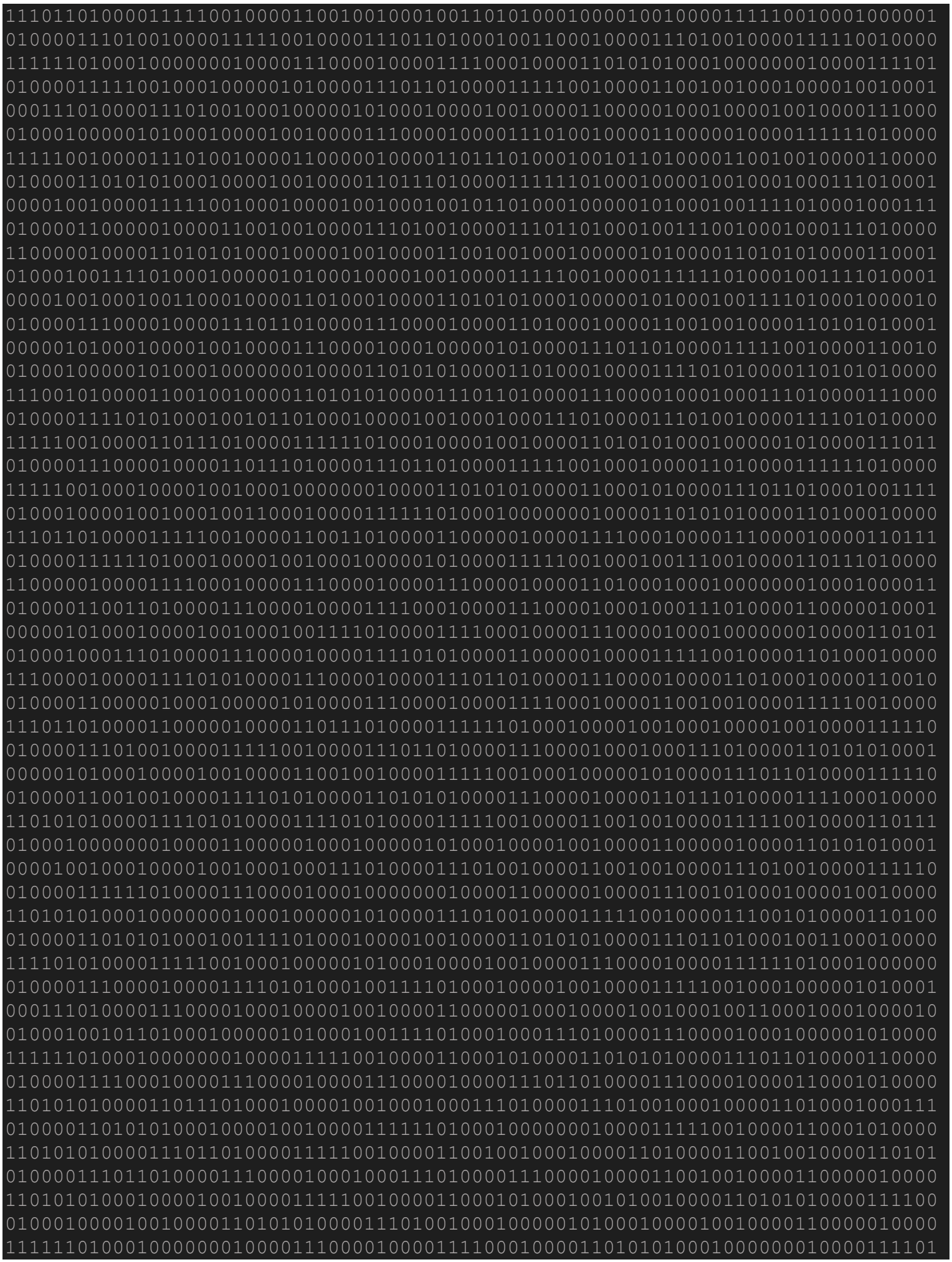
```
                reg_x_temp[j] = reg_x_temp_two[j-1]
                j = j + 1
            reg_x_temp[0] = new

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

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

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

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

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

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

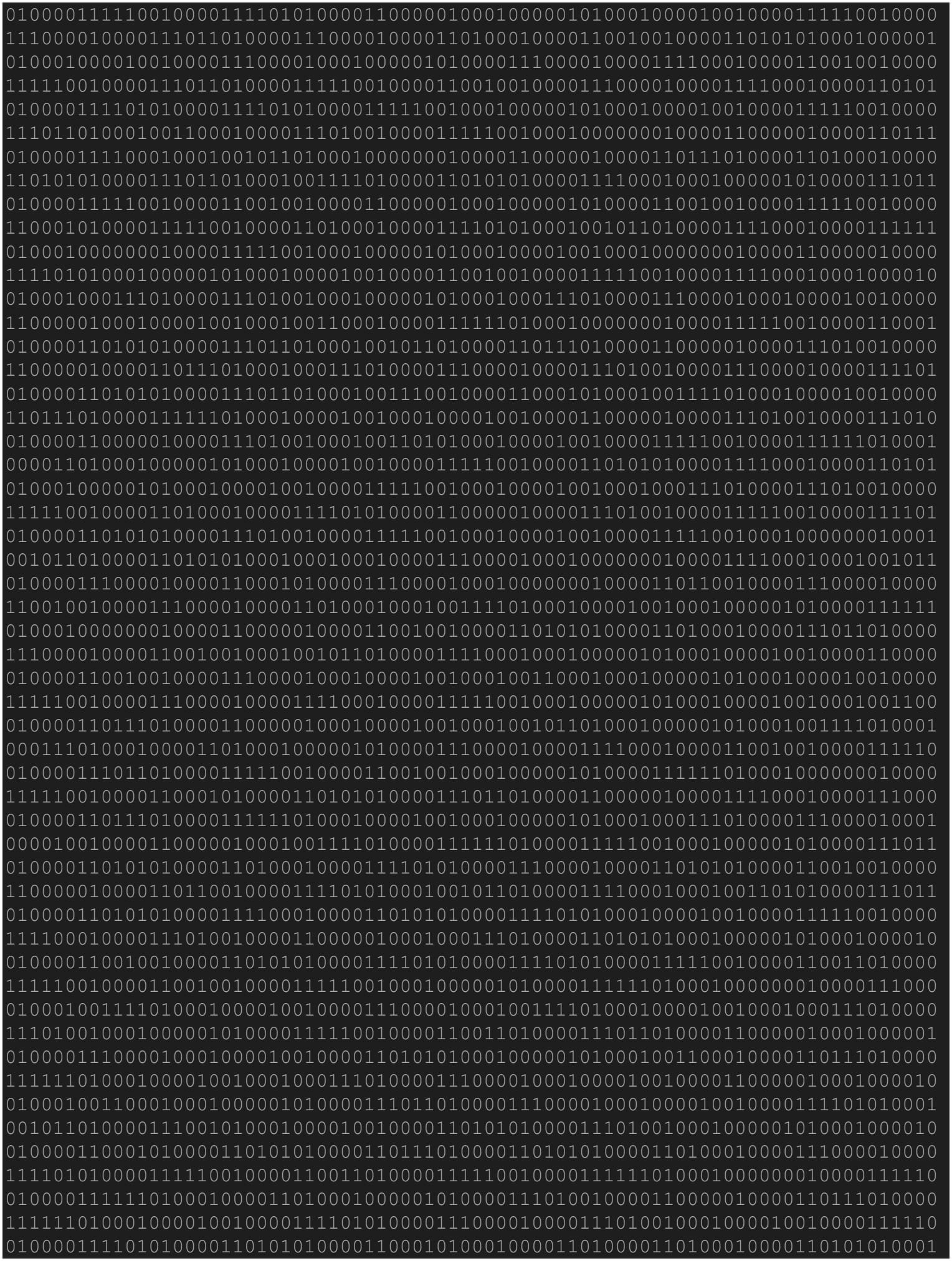
```
    while(i < len(cipher)):
        binary.insert(i, int(cipher[i]))
        s = s + str(binary[i] ^ keystream[i])
        i = i + 1
    return convert_binary_to_str(str(s))

key = '0101001000011010110001110001100100101001000000110111111010110111'
set_key(key)


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

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

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

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

Тестирование:

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

A5/2:
Ключ: 0101001000011010110001110001100100101001000000110111111010110111
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
010000110010010001000000010000110101010000111100010001001111010000110111010000
111111010001000010010000111111010001000000010000111000010000111011010000111000
010000110010010001001011010000111000010000111110010001000010010000111011010000
111000010000110010010001001011010000111101010000110101010000110110010000110100
010001000011010001000010010001000011101000011010101000011101101000011111001000
011001001000011010101000011101001000011000001000100001001000100011101000011101 0

Расшифрованный текст:
время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
```





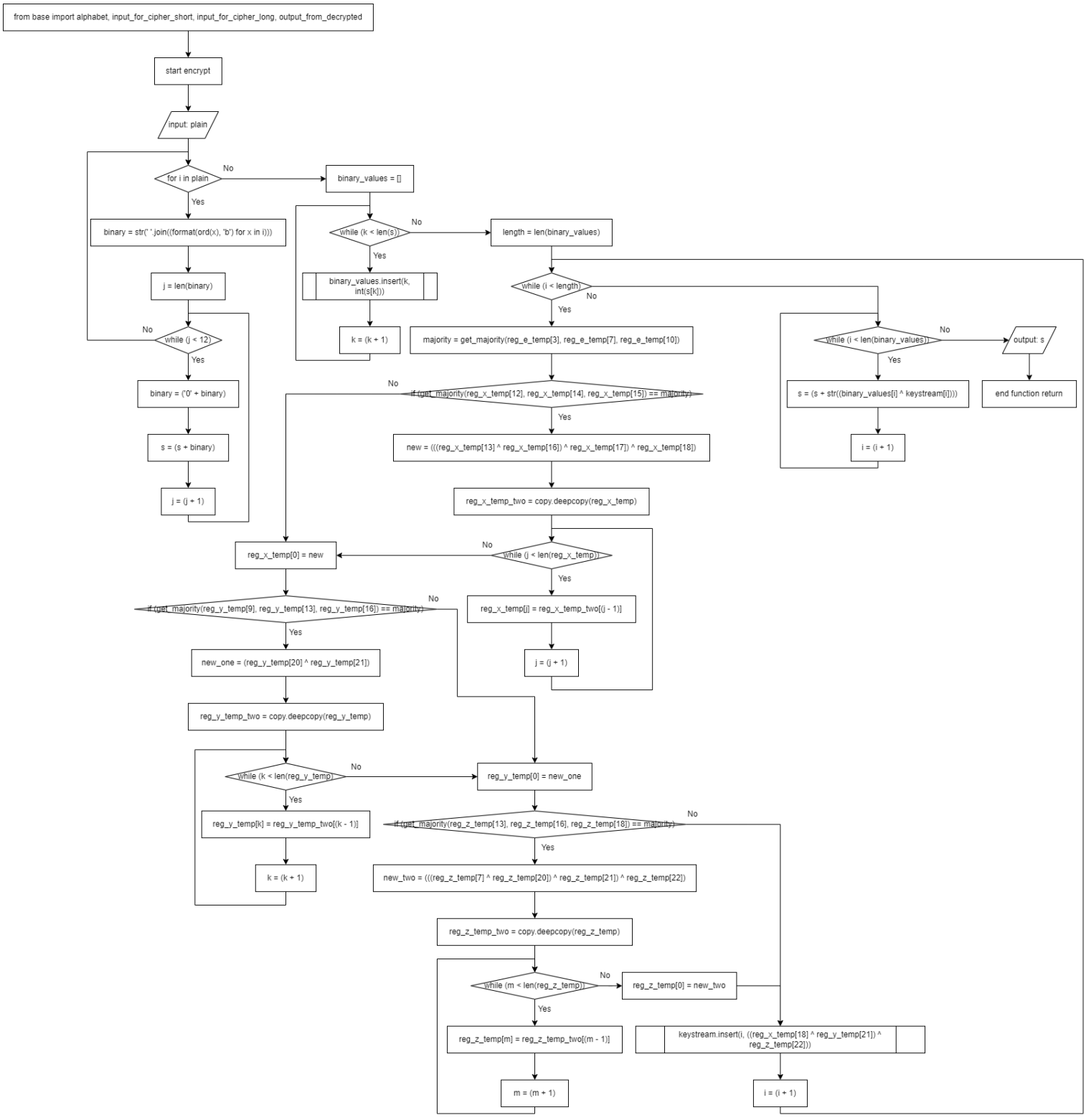


```
00001001000100001001000100011101000011101001000011110101000011111001000011000
101000011111001000011101101000100110001000100100001000011100001000011110101000
100000101000100001001000011001001000100001101000011110101000100001101000011011
001000011110101000011000001000100011001000011010101000011110101000011000001000
001101110100001100000100010000100100010010110100010000010100010011110100010001
110100010000110100001101110100001111010100001100000100001110100100001111100100
000110010010000110001010000110101010000110111010000111111010001000000001000011
111001000011000101000011010101000011101101000011111001000011001001000100001001
00010001110100001110101000011101001000011110101000011111100100001100010100001
10101010000111011010000111110010000110010010001000010010001
00011101000011101 0
```

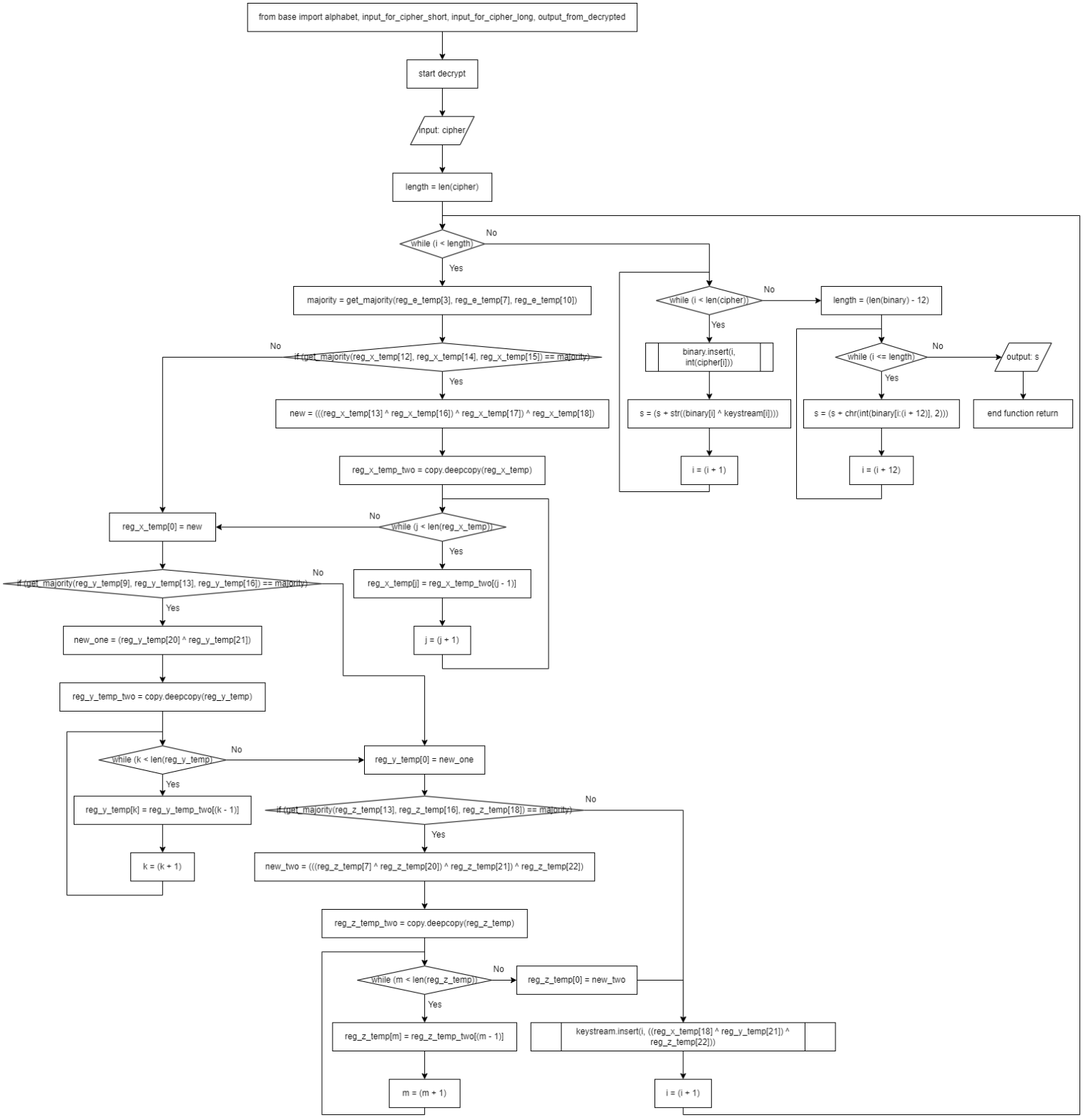
Расшифрованный текст:
вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходящ
ийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпубликац
ий.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.номож
ноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднукартин
у.текстнатысячусимволовэтосколькопримернослов.статистикапоказывает,чтотысячавк
лючаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблятьпредл
огами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеизменнов
озрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиилибез.учет
пробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименностолькоразмы
разделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,таккакэтоп
устоеместо.однаконекоторыефирмыибиржививидятсправедливымставитьстоимостьзатысячу
символовспробелами,считаяпоследниеважнымэлементомкачественноговосприятия.согла
ситесь,читатьслитныйтекстбезединогопропуска,никтонебудет.нобольшинствунужнацен
азатысячузнаковбезпробелов.

Блок-схема:

Шифрование:



Дешифрование:



Блок G: КОМБИНАЦИОННЫЕ ШИФРЫ

17.МАГМА

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

Код программы:

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

pi0 = [12, 4, 6, 2, 10, 5, 11, 9, 14, 8, 13, 7, 0, 3, 15, 1]
pi1 = [6, 8, 2, 3, 9, 10, 5, 12, 1, 14, 4, 7, 11, 13, 0, 15]
pi2 = [11, 3, 5, 8, 2, 15, 10, 13, 14, 1, 7, 4, 12, 9, 6, 0]
pi3 = [12, 8, 2, 1, 13, 4, 15, 6, 7, 0, 10, 5, 3, 14, 9, 11]
pi4 = [7, 15, 5, 10, 8, 1, 6, 13, 0, 9, 3, 14, 11, 4, 2, 12]
pi5 = [5, 13, 15, 6, 9, 2, 12, 10, 11, 7, 8, 1, 4, 3, 14, 0]
pi6 = [8, 14, 2, 5, 6, 9, 1, 12, 15, 4, 11, 0, 13, 10, 3, 7]
pi7 = [1, 7, 14, 13, 0, 5, 8, 3, 4, 15, 10, 6, 9, 12, 11, 2]

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

MASK32 = 2 ** 32 - 1

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

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

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

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

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

def magma_key_schedule(k):
    keys = []
    for i in reversed(range(8)):
        keys.append((k >> (32 * i)) & MASK32)
    for i in range(8):
        keys.append(keys[i])
```

```
    for i in range(8):
        keys.append(keys[i])
    for i in reversed(range(8)):
        keys.append(keys[i])
    return keys

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

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

key = int('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))}
''')
```

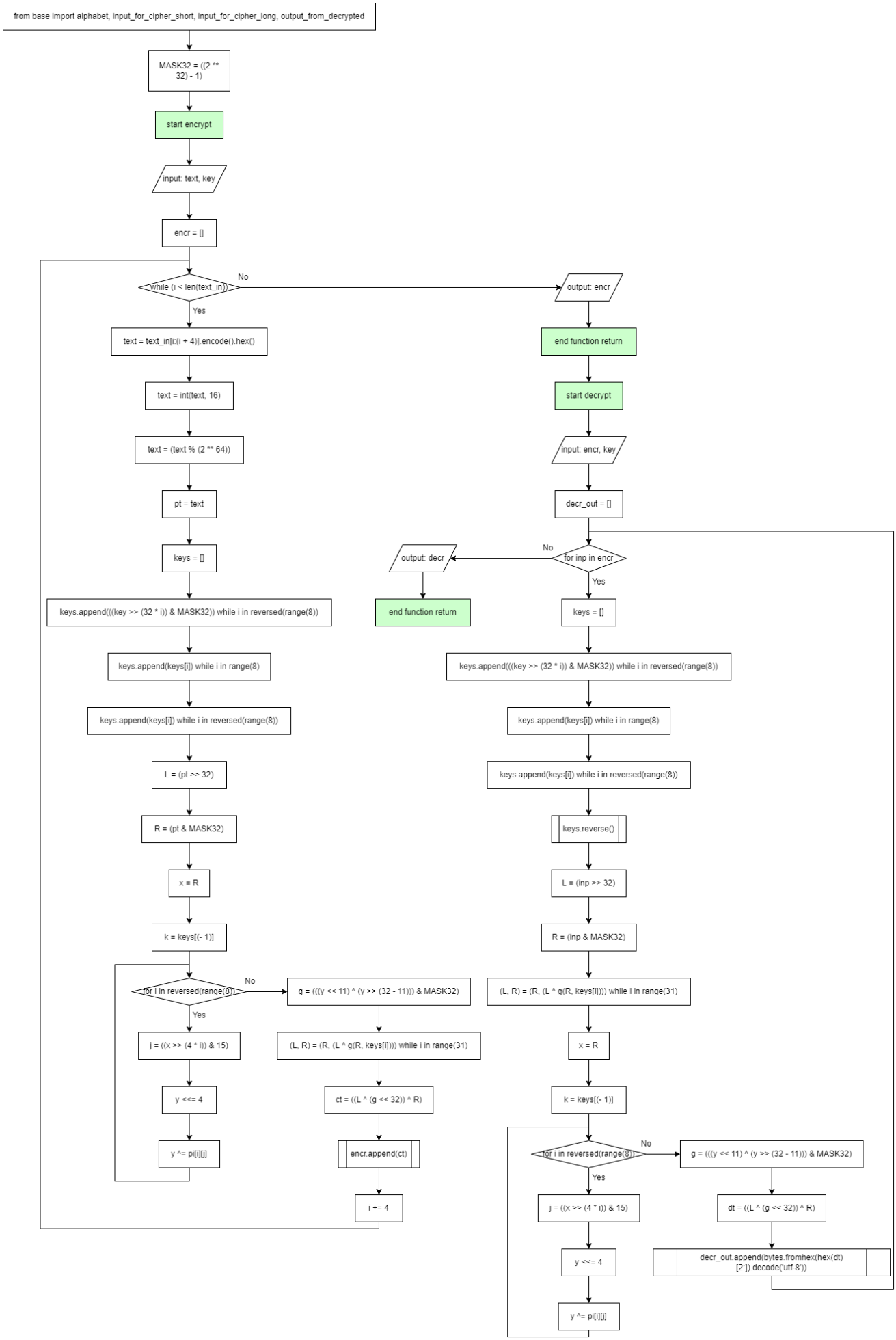
Тестирование:

```
/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:
[83438756020388080 58, 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, 93295469440628314 70,
34182447647981437 90, 15295462055836379806, 51556803592830345 01,
94519373805012329 12, 73282205569576205 99, 36149370742366765 57,
45891814995130458 79, 13803166415433109120, 18278897981036660435,
40199369863182153 94, 12394731635219602309, 54162488580339238 16,
46323282895002593 3, 15989759834534178049, 13360939273674915028,
85286085552919060 00, 10293453731533650333, 13238975415012678998,
12616809652721693 68, 10483829524585249195, 15703560336104773581,
29286127887899747 86, 12370719347969566168, 15498018845651124401,
79752020056154359 22, 10016402103949689 31, 94823908768763094 60,
10306886503249707528, 34323711200831045 76, 13284723522159984134,
47329579363084845 48, 18145150074698797421, 76659126580164409 55,
17186319045640305491, 13257185717927971463, 39679573280287354 47,
63030451061339715 10, 11637202970910093 42, 23466997127566948 09,
10356924971308547229, 13990735642799502546, 16656837303721777190,
36209708847117289 77, 94434872926303641 51, 76910490946820269 09,
91611207317717494 4, 15797466941996902229, 16777509928585489639,
63031214702137987 11, 11674884899395813190, 18046574429007017128,
13077766683806041775, 65692148579544394 28, 13257185717927971463,
39679573280287354 47, 63030451061339715 10, 18131068278186858513,
16002418093945299027, 29081148687780745 20, 11794392602089101467,
11996918218642771 91, 30769383504769500 06, 13450817471486594379,
10728741519359495825, 92691896824800915 97, 14111778268298096419,
20578963828445017 04, 10293453731533650333, 17204212421686848739,
53662062268688007 02, 82484669974485367 22, 10088810565957275850,
22543550902437422 57, 83718968843992981 33, 10985518332543285851,
17941711041904849701, 77402002079432793 54, 10750419551538638209,
26292977765199590 89, 22257420270936085 02, 28724497422719375 1,
17712479779128577940, 92222969970569946 15, 97378850006718156 0,
33935424538133483 9, 10641483480207737426, 16269104980183698483,
16250235473790191753, 27006887292698816 30, 84116078843487540 50,
76364480487959567 98, 69274644265231551 5, 11343070527907873916,
62952192059935043 0, 14623128418242633989, 57052476339081059 59,
17910568456840661856, 16201319981229519019, 13769298882463179721,
74211540778054854 56, 93121838771575555 60, 76910490946820269 09,
91611207317717494 4, 15135398776342705892, 15289260373132081671,
71263790601415682 92, 17250369459672726 73, 17688755856354297123,
13811350864339364 3, 14127475102588350875, 41510163563411591 75,
98414674873937416 36, 31879593274959246 97, 15728081324422447459,
16055605958376054530, 35722660140173463 59, 32238270005876804 74,
82537335001663112 51, 91405085422878099 51, 17900326362396929 33,
14268543881935307321, 73424823037920083 83, 17677570963908781400,
13355264069952555938, 88597409000583480, 17257030184527390406,
12715322944275189718, 43113056607886706 62, 70084620566287577 49,
57052476339081059 59, 78047154577701729 5, 71813374295849181 54,
17602858510462705198, 14738061815484779755, 54029931001616126 4,
10182683375894303875, 80263432045513379 82, 17660378940581534682,
13825198160625090606, 47205882698961406 16, 57305274094012220 56,
17941293450735671408, 43423663519841470 86, 53132764481581653 79,
17450241683162283 46, 25281365199649886 65, 35477558118573890 63,
10306886503249707528, 45264948651592282 42, 73424823037920083 83,
29081148687780745 20, 74716800385967772 70, 16412799349039310306,
17456146687817481297, 28943214759462716 48, 69231581617379714 79,

```
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, 1512703795048864 41, 10183946694284410805,
580256645846205130, 5356452764985505955, 1949584005305844760,
5054509616596506897, 8065619470667740 48, 14232614625982377749,
4486402459089481420, 6142067374720541917, 11774712973292423210,
10305629430321463988, 3547755811857389063, 10306886503249707528,
15489555906471766183, 15441006425873291865, 2659805083408552087,
12582666581260422456, 541882649396588128, 4886609086442669621,
5561196856767141965, 7731325888395147745, 10966269065365886 42,
763947604362793674, 12798794780496052712, 82992425697894351 68,
67660617073868317 62, 89715285509582270 67, 11254181301937016096,
41067934786911701 80, 58123053744651110 22, 14122431208396972645,
87568808396852895 05, 18401452094994769 46, 17204212421686848739,
12702832949266190125, 12312436585117905032, 2731161190011158263,
47205882698961406 16, 13119149454917978330, 98221088514466533 48,
40856774283330940 07, 12601379616160262740, 10265087333462641478,
87709561803165343 33, 27462124300767568 19, 10780803050552638165,
19061242771921034 47, 83187902879887032 11, 94519373805012329 12,
71540105892017162 26, 65932387259159398 56, 16381960257093506421,
16805496448556233763, 13257185717927971463, 35342388718972839 87,
18436104050552264826, 32371662915203870 28, 14738061815484779755,
540299310016161264, 10585345143535530769]

Расшифрованный текст:
вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходящ
ийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпубликац
ий.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.номож
ноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднукартин
у.текстнатысячусимволовэтосколькопримернослов.статистикапоказывает,чтотысячавк
лючаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблятьпредл
огами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеизменнов
озрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиилибез.учет
пробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименностолькоразмы
разделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,таккакэтоп
устоеместо.однаконекоторыефирмыибирживидятсправедливымставитьстоимостьзатысячу
символовспробелами,считаяпоследниеважнымэлементомкачественноговосприятия.согла
ситесь,читатьслитныйтекстбезединогопропуска,никтонебудет.небольшинствунужнацен
азатысячузнаковбезпробелов.
```

Блок-схема:


from base import alphabet, input_for_cipher_short, input_for_cipher_long, output_from_decrypted
MASK32 = ((2 ** 32) - 1)
start encrypt
input: text, key
encr = []
while (i < len(text_in))
No
Yes
output: encr
text = text_in[i:(i + 4)].encode().hex()
end function return
text = int(text, 16)
start decrypt
text = (text % (2 ** 64))
input: encr, key
pt = text
decr_out = []
keys = []
for inp in encr
No
output: decr
Yes
keys.append(((key >> (32 * i)) & MASK32)) while i in reversed(range(8))
end function return
keys = []
keys.append(keys[i]) while i in range(8)
keys.append(((key >> (32 * i)) & MASK32)) while i in reversed(range(8))
keys.append(keys[i]) while i in reversed(range(8))
keys.append(keys[i]) while i in range(8)
L = (pt >> 32)
keys.append(keys[i]) while i in reversed(range(8))
R = (pt & MASK32)
keys.reverse()
x = R
L = (inp >> 32)
k = keys[(- 1)]
R = (inp & MASK32)
for i in reversed(range(8))
No
g = (((y << 11) ^ (y >> (32 - 11))) & MASK32)
(L, R) = (R, (L ^ g(R, keys[i]))) while i in range(31)
Yes
j = ((x >> (4 * i)) & 15)
(L, R) = (R, (L ^ g(R, keys[i]))) while i in range(31)
x = R
y <<= 4
ct = ((L ^ (g << 32)) ^ R)
k = keys[(- 1)]
y ^= pi[i][j]
encr.append(ct)
for i in reversed(range(8))
No
g = (((y << 11) ^ (y >> (32 - 11))) & MASK32)
i += 4
Yes
j = ((x >> (4 * i)) & 15)
dt = ((L ^ (g << 32)) ^ R)
y <<= 4
decr_out.append(bytes.fromhex(hex(dt)[2:]).decode('utf-8'))
y ^= pi[i][j]

БЛОК H: АСИММЕТРИЧНЫЕ ШИФРЫ

21.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 Ci 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:

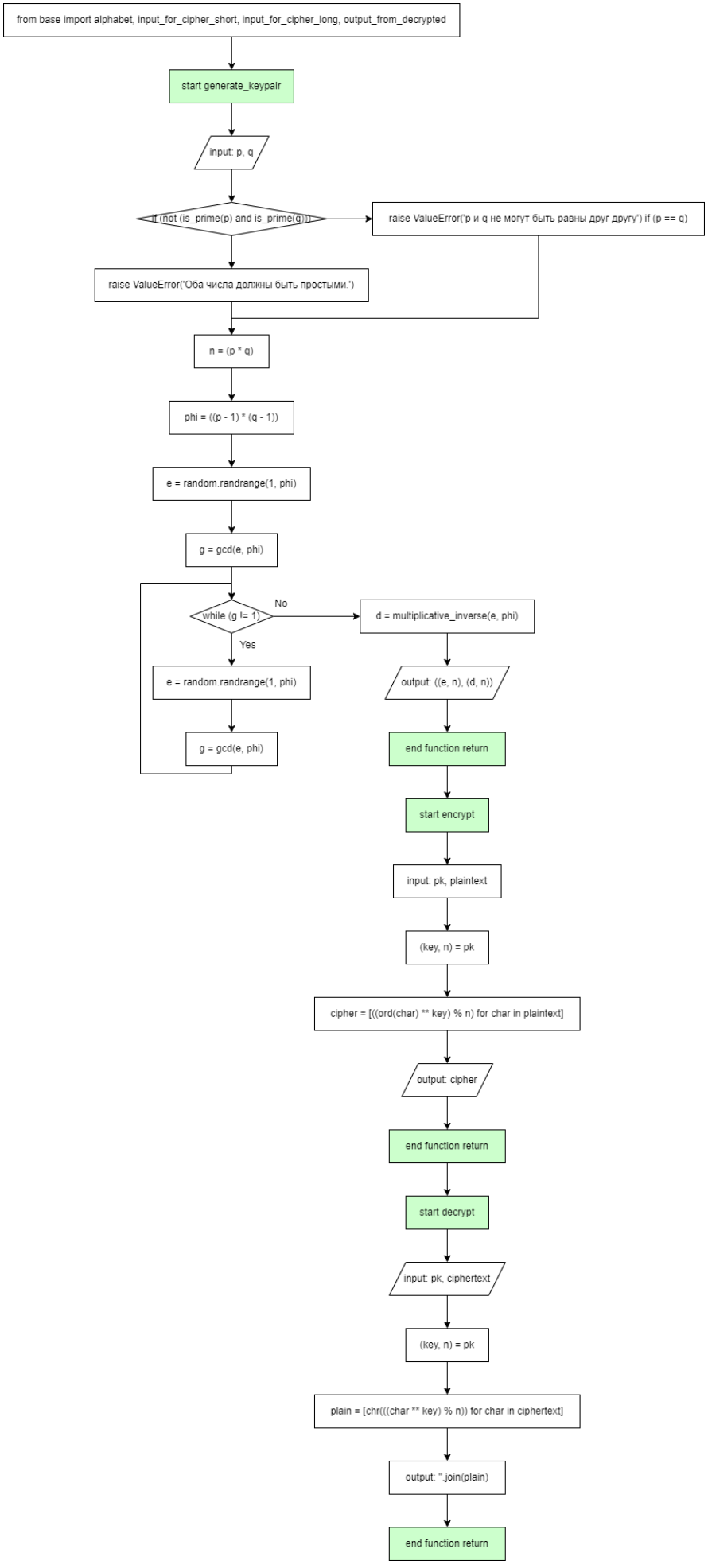
```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab08_21_rsa.py
↵

RSA:
Ключ:
p=107 q=109
Публичный: (5441, 11663)
Приватный: (4025, 11663)
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
759271718847938599013012565910028565971711881804188759210654188633810028180418
875921065484778847319610754100291002864568847180463387592884725481001810028645
62548

Расшифрованный текст:
время,приливыиотливынеждутчеловека.

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

```
804439084776338217910028188565971711888477990110028633821796456188100281001810
028439010028106542179990164561882179565971716338310788471804100189385188188180
418831078847301210028645625481002964568847100285659717163383107884718046338759
210029759288471804188645618875921001888471002863383107562788479385100288847254
821791002810018565971711889385884771718477633884771001821791002863381881804188
107547592884721791002818821791889385759263381804633875921889385884784778477633
821791002863381804439025486338717110018301293851065471711001830121075488471804
990188479385217918046338759210018217975926338310763381075484771065493855659717
163382179100287171100188477217910028759263389385100286456254821796456188100281
001810028439056597171633831078847180410654301210018254810018301264561882548188
847788471804115193107990110028301256591002810028100182548254810018254826561002
863385659100292179100286338884793858847217910028633810028645625486338107548477
100182548633884778847254863381002863387171106548847852518871719385106541883107
188717131961887592188107549901100282179565971711001875928847107541804188759210
654938521791002810018759218810028439021791002863381889385633821791002843903012
100181002810654217999016456100292179188938575926338180463387592217956597171633
831078847180410018938518830125659100282179645618810028100189901565963382179180
488471075484771888847759210018319684771065493852656180488479385884784771002863
389385254810018645688472179100287592884784778477633865336338759263382179565971
711889901100281889901100286456254821796338653318041001821791881002888472179439
030125659100286456188100281001810028439021791804188100288477106546568100288847
254821791002831078847301288471075418884776338653363385659717163385659100292179
254810018301256591002884771882548100286338847788473107100291075488471002810028
645625488477633831076338180443904131188847721791002875921002984771002931968477
100189467884784771001830121001810028106542179990164561002930128477100182548633
875923107884730125659717163383107884718046338759210028645625 48
```

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:


from base import alphabet, input_for_cipher_short, input_for_cipher_long, output_from_decrypted
start generate_keypair
input: p, q
if (not (is_prime(p) and is_prime(q)))
raise ValueError('p и q не могут быть равны друг другу') if (p == q)
raise ValueError('Оба числа должны быть простыми.')
n = (p * q)
phi = ((p - 1) * (q - 1))
e = random.randrange(1, phi)
g = gcd(e, phi)
while (g != 1)
No
Yes
d = multiplicative_inverse(e, phi)
e = random.randrange(1, phi)
g = gcd(e, phi)
output: ((e, n), (d, n))
end function return
start encrypt
input: pk, plaintext
(key, n) = pk
cipher = [((ord(char) ** key) % n) for char in plaintext]
output: cipher
end function return
start decrypt
input: pk, ciphertext
(key, n) = pk
plain = [chr(((char ** key) % n)) for char in ciphertext]
output: ''.join(plain)
end function return

22.Elgamal

Пусть M — 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 p (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) T.e. решается сравнение относительно M: Для этого надо избавиться от знаменателя: умножаем левую и правую часть сравнения на ax и решаем полученное сравнение относительно M, пользуясь расширенным алгоритмом Евклида: ax M ≡ 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,
```

45460722162973005037141629215899 2, 46393684744675809244993416382643 4,
45799981283592206567269551821988 0, 46054425624056607714865493776554 6,
46012018233979207523599503450793 5, 46012018233979207523599503450793 5,
46605721695062810201323368011448 9, 46351277354598409053727426056882 3,
46096833014134007906131484102315 7, 46266462574443608671195445405360 1,
45503129553050405228407619541660 3, 45927203453824407141067522799271 3,
45799981283592206567269551821988 0, 45884796063747006949801532473510 2,
45460722162973005037141629215899 2, 46393684744675809244993416382643 4,
45799981283592206567269551821988 0, 45842388673669606758535542147749 1,
46224055184366208479929455079599 0, 46436092134753209436259406708404 5,
45884796063747006949801532473510 2, 45545536943127805419673609867421 4,
46224055184366208479929455079599 0, 45460722162973005037141629215899 2,
45884796063747006949801532473510 2, 46054425624056607714865493776554 6,
45969610843901807332333513125032 4, 46224055184366208479929455079599 0,
45672759113360005993471580844704 7, 45884796063747006949801532473510 2,
46181647794288808288663464753837 9, 46224055184366208479929455079599 0,
45672759113360005993471580844704 7, 46139240404211408097397474428076 8,
45672759113360005993471580844704 7, 45630351723282605802205590518943 6,
45884796063747006949801532473510 2, 46054425624056607714865493776554 6,
45503129553050405228407619541660 3, 46605721695062810201323368011448 9,
45545536943127805419673609867421 4, 45460722162973005037141629215899 2,
45672759113360005993471580844704 7, 46224055184366208479929455079599 0,
45503129553050405228407619541660 3, 46054425624056607714865493776554 6,
45927203453824407141067522799271 3, 45672759113360005993471580844704 7,
45672759113360005993471580844704 7, 45630351723282605802205590518943 6,
45545536943127805419673609867421 4, 46266462574443608671195445405360 1,
46351277354598409053727426056882 3, 45799981283592206567269551821988 0,
45927203453824407141067522799271 3, 45799981283592206567269551821988 0,
46224055184366208479929455079599 0, 46139240404211408097397474428076 8,
46860166035527211348919309966015 5, 46351277354598409053727426056882 3,
45460722162973005037141629215899 2, 45503129553050405228407619541660 3,
45757573893514806376003561496226 9, 45460722162973005037141629215899 2,
46393684744675809244993416382643 4, 45672759113360005993471580844704 7,
45545536943127805419673609867421 4, 45799981283592206567269551821988 0,
46054425624056607714865493776554 6, 45503129553050405228407619541660 3,
46605721695062810201323368011448 9, 46436092134753209436259406708404 5,
46012018233979207523599503450793 5, 46054425624056607714865493776554 6,
46054425624056607714865493776554 6, 45630351723282605802205590518943 6,
45799981283592206567269551821988 0, 46012018233979207523599503450793 5,
46096833014134007906131484102315 7, 46054425624056607714865493776554 6,
45630351723282605802205590518943 6, 45757573893514806376003561496226 9,
45460722162973005037141629215899 2, 45587944333205205610939600193182 5,
46054425624056607714865493776554 6, 45927203453824407141067522799271 3,
46054425624056607714865493776554 6, 45545536943127805419673609867421 4,
46054425624056607714865493776554 6, 45884796063747006949801532473510 2,
46224055184366208479929455079599 0, 46436092134753209436259406708404 5,
45884796063747006949801532473510 2, 46012018233979207523599503450793 5,
46054425624056607714865493776554 6, 45969610843901807332333513125032 4,
46054425624056607714865493776554 6, 45715166503437406184737571170465 8,
46012018233979207523599503450793 5, 46054425624056607714865493776554 6,
45799981283592206567269551821988 0, 45503129553050405228407619541660 3,
45672759113360005993471580844704 7, 45757573893514806376003561496226 9,
46012018233979207523599503450793 5, 45672759113360005993471580844704 7,

```
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,
```

```
4605442562405660771486549377655 46, 4609683301413400790613148410231 57,
4613924040421140809739747442807 68, 4579998128359220656726955182198 80,
4596961084390180733233351312503 24, 4567275911336000599347158084470 47,
4613924040421140809739747442807 68, 4601201823397920752359950345079 35,
4605442562405660771486549377655 46, 4618164779428880828866346475383 79,
4592720345382440714106752279927 13, 4605442562405660771486549377655 46,
4554553694312780541967360986742 14, 4622405518436620847992945507959 90,
4643609213475320943625940670840 45, 4588479606374700694980153247351 02,
4618164779428880828866346475383 79, 4622405518436620847992945507959 90,
4546072216297300503714162921589 92, 4622405518436620847992945507959 90,
4579998128359220656726955182198 80, 4618164779428880828866346475383 79,
4622405518436620847992945507959 90, 4579998128359220656726955182198 80,
4588479606374700694980153247351 02, 4546072216297300503714162921589 92,
4609683301413400790613148410231 57, 4605442562405660771486549377655 46,
4588479606374700694980153247351 02, 4546072216297300503714162921589 92,
4575757389351480637600356149622 69, 4660572169506281020132336801144 89,
4554553694312780541967360986742 14, 4546072216297300503714162921589 92,
4567275911336000599347158084470 47, 4622405518436620847992945507959 90,
4575757389351480637600356149622 69, 4609683301413400790613148410231 57,
4622405518436620847992945507959 90, 4643609213475320943625940670840 45,
4622405518436620847992945507959 90, 4605442562405660771486549377655 46,
4622405518436620847992945507959 90, 4660572169506281020132336801144 89,
4618164779428880828866346475383 79, 4677535125537241096638732931449 33,
4643609213475320943625940670840 45, 4546072216297300503714162921589 92,
4554553694312780541967360986742 14, 4588479606374700694980153247351 02,
4592720345382440714106752279927 13, 4673294386529501077512133898873 22,
4643609213475320943625940670840 45, 4546072216297300503714162921589 92,
4567275911336000599347158084470 47, 4622405518436620847992945507959 90,
4554553694312780541967360986742 14, 4618164779428880828866346475383 79,
4567275911336000599347158084470 47, 4550312955305040522840761954166 03,
4677535125537241096638732931449 33, 4618164779428880828866346475383 79,
4622405518436620847992945507959 90, 4605442562405660771486549377655 46,
4609683301413400790613148410231 57, 4677535125537241096638732931449 33,
4622405518436620847992945507959 90, 4664812908514021039258935833721 00,
4563035172328260580220559051894 36, 4567275911336000599347158084470 47,
4618164779428880828866346475383 79, 4677535125537241096638732931449 33,
4622405518436620847992945507959 90, 4579998128359220656726955182198 80,
4592720345382440714106752279927 13, 4579998128359220656726955182198 80,
4563035172328260580220559051894 36, 4554553694312780541967360986742 14,
4567275911336000599347158084470 47, 4618164779428880828866346475383 79,
4622405518436620847992945507959 90, 4579998128359220656726955182198 80,
4618164779428880828866346475383 79, 4592720345382440714106752279927 13,
4605442562405660771486549377655 46, 4554553694312780541967360986742 14,
4618164779428880828866346475383 79, 4613924040421140809739747442807 68,
4567275911336000599347158084470 47, 4563035172328260580220559051894 36,
4601201823397920752359950345079 35, 4567275911336000599347158084470 47,
4584238867366960675853554214774 91, 4554553694312780541967360986742 14,
4567275911336000599347158084470 47, 4592720345382440714106752279927 13,
4579998128359220656726955182198 80, 4643609213475320943625940670840 45,
4579998128359220656726955182198 80, 4601201823397920752359950345079 35,
4660572169506281020132336801144 89, 4622405518436620847992945507959 90,
4643609213475320943625940670840 45, 4588479606374700694980153247351 02,
4601201823397920752359950345079 35, 4605442562405660771486549377655 46,
```

```
45757573893514806376003561496226 9, 46096833014134007906131484102315 7,
46224055184366208479929455079599 0, 45672759113360005993471580844704 7,
46181647794288808288663464753837 9, 45927203453824407141067522799271 3,
45799981283592206567269551821988 0, 45757573893514806376003561496226 9,
45927203453824407141067522799271 3, 46054425624056607714865493776554 6,
46266462574443608671195445405360 1, 46096833014134007906131484102315 7,
46054425624056607714865493776554 6, 46224055184366208479929455079599 0,
46139240404211408097397474428076 8, 45672759113360005993471580844704 7,
45503129553050405228407619541660 3, 45927203453824407141067522799271 3,
46775351255372410966387329314493 3, 46224055184366208479929455079599 0,
46648129085140210392589358337210 0, 46096833014134007906131484102315 7,
46139240404211408097397474428076 8, 45672759113360005993471580844704 7,
45630351723282605802205590518943 6, 45927203453824407141067522799271 3,
46054425624056607714865493776554 6, 45587944333205205610939600193182 5,
45460722162973005037141629215899 2, 45969610843901807332333513125032 4,
45799981283592206567269551821988 0, 45757573893514806376003561496226 9,
46096833014134007906131484102315 7, 46224055184366208479929455079599 0,
46181647794288808288663464753837 9, 46054425624056607714865493776554 6,
46732943865295010775121338988732 2, 45757573893514806376003561496226 9,
45460722162973005037141629215899 2, 45969610843901807332333513125032 4,
45799981283592206567269551821988 0, 45799981283592206567269551821988 0,
45630351723282605802205590518943 6, 46139240404211408097397474428076 8,
46266462574443608671195445405360 1, 45587944333205205610939600193182 5,
45799981283592206567269551821988 0, 45969610843901807332333513125032 4,
45799981283592206567269551821988 0, 46436092134753209436259406708404 5,
45460722162973005037141629215899 2, 46181647794288808288663464753837 9,
46224055184366208479929455079599 0, 46775351255372410966387329314493 3,
45969610843901807332333513125032 4, 45799981283592206567269551821988 0,
46139240404211408097397474428076 8, 45672759113360005993471580844704 7,
46436092134753209436259406708404 5, 45799981283592206567269551821988 0,
46012018233979207523599503450793 5, 45460722162973005037141629215899 2,
46054425624056607714865493776554 6, 45630351723282605802205590518943 6,
45799981283592206567269551821988 0, 46012018233979207523599503450793 5,
45799981283592206567269551821988 0, 45927203453824407141067522799271 3,
45799981283592206567269551821988 0, 45630351723282605802205590518943 6,
45545536943127805419673609867421 4, 45460722162973005037141629215899 2,
46181647794288808288663464753837 9, 45799981283592206567269551821988 0,
45969610843901807332333513125032 4, 45545536943127805419673609867421 4,
46054425624056607714865493776554 6, 45927203453824407141067522799271 3,
45460722162973005037141629215899 2, 45757573893514806376003561496226 9,
46096833014134007906131484102315 7, 46224055184366208479929455079599 0,
46224055184366208479929455079599 0, 46054425624056607714865493776554 6,
45884796063747006949801532473510 2, 46054425624056607714865493776554 6,
45927203453824407141067522799271 3, 45799981283592206567269551821988 0,
46436092134753209436259406708404 5, 45672759113360005993471580844704 7,
46181647794288808288663464753837 9, 46224055184366208479929455079599 0,
45545536943127805419673609867421 4, 46054425624056607714865493776554 6,
46181647794288808288663464753837 9, 45927203453824407141067522799271 3,
46054425624056607714865493776554 6, 45545536943127805419673609867421 4,
46012018233979207523599503450793 5, 45672759113360005993471580844704 7,
45799981283592206567269551821988 0, 45757573893514806376003561496226 9,
45969610843901807332333513125032 4, 45672759113360005993471580844704 7,
46012018233979207523599503450793 5, 46012018233979207523599503450793 5,
```

```
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,
```

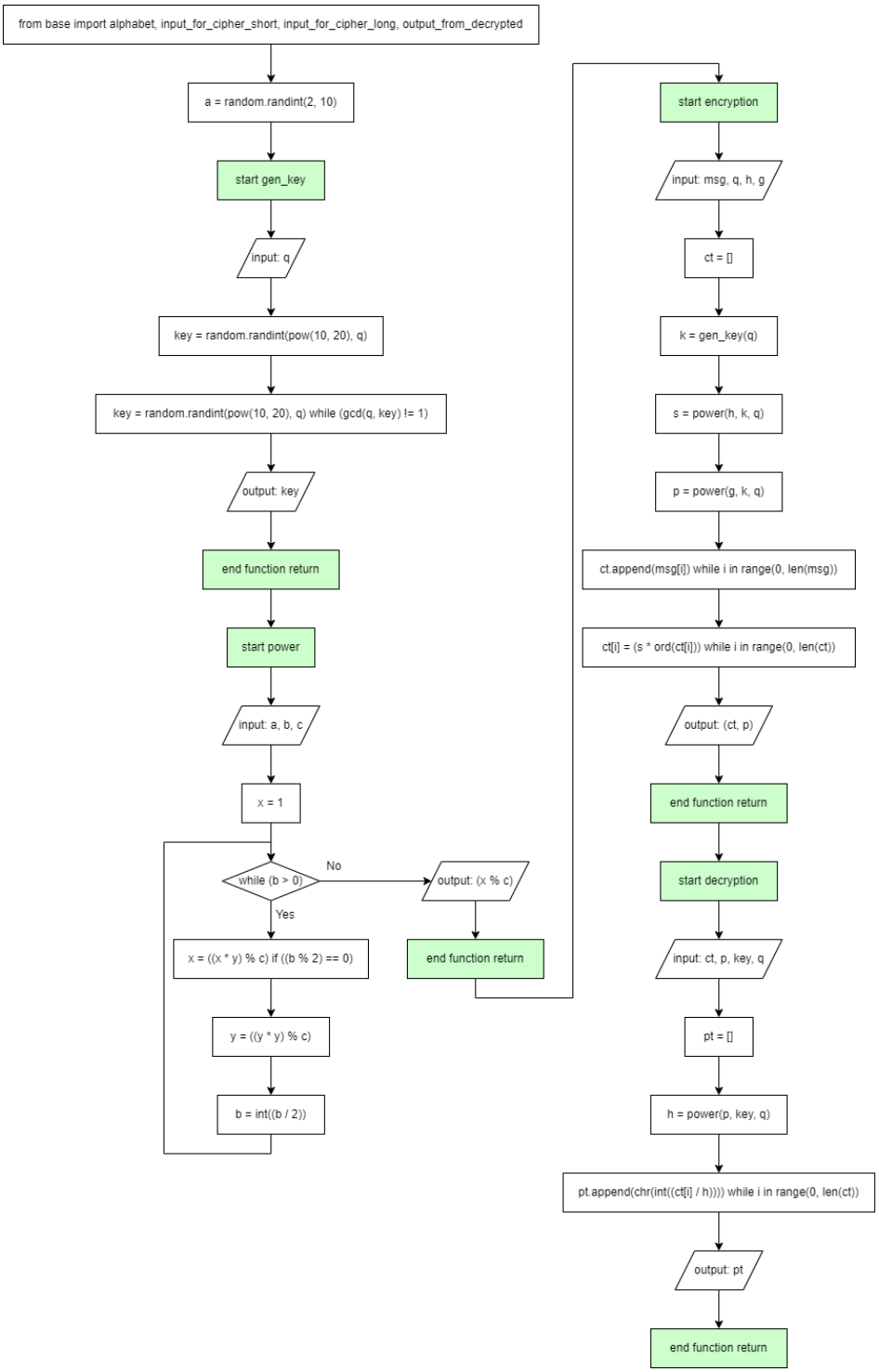
```
456727591133600059934715808447047, 459272034538244071410675227992713,
466057216950628102013233680114489, 457575738935148063760035614962269,
454607221629730050371416292158992, 458847960637470069498015324735102,
454607221629730050371416292158992, 457575738935148063760035614962269,
464360921347532094362594067084045, 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,
```

```
46224055184366208479929455079599O, 46605721695062810201323368011448 9,
46181647794288808288663464753837 9, 46775351255372410966387329314493 3,
46436092134753209436259406708404 5, 46266462574443608671195445405360 1,
46181647794288808288663464753837 9, 45799981283592206567269551821988 0,
45969610843901807332333513125032 4, 45545536943127805419673609867421 4,
46054425624056607714865493776554 6, 45927203453824407141067522799271 3,
46054425624056607714865493776554 6, 45545536943127805419673609867421 4,
46181647794288808288663464753837 9, 46096833014134007906131484102315 7,
46139240404211408097397474428076 8, 46054425624056607714865493776554 6,
45503129553050405228407619541660 3, 45672759113360005993471580844704 7,
45927203453824407141067522799271 3, 45460722162973005037141629215899 2,
45969610843901807332333513125032 4, 45799981283592206567269551821988 0,
45757573893514806376003561496226 9, 46096833014134007906131484102315 7,
46224055184366208479929455079599O, 46181647794288808288663464753837 9,
46436092134753209436259406708404 5, 45799981283592206567269551821988 0,
46224055184366208479929455079599O, 45460722162973005037141629215899 2,
46775351255372410966387329314493 3, 46096833014134007906131484102315 7,
46054425624056607714865493776554 6, 46181647794288808288663464753837 9,
45927203453824407141067522799271 3, 45672759113360005993471580844704 7,
45630351723282605802205590518943 6, 46012018233979207523599503450793 5,
45799981283592206567269551821988 0, 45672759113360005993471580844704 7,
45545536943127805419673609867421 4, 45460722162973005037141629215899 2,
45715166503437406184737571170465 8, 46012018233979207523599503450793 5,
46605721695062810201323368011448 9, 45969610843901807332333513125032 4,
46690536475217610583855348662971 1, 45927203453824407141067522799271 3,
45672759113360005993471580844704 7, 45969610843901807332333513125032 4,
45672759113360005993471580844704 7, 46012018233979207523599503450793 5,
46224055184366208479929455079599O, 46054425624056607714865493776554 6,
45969610843901807332333513125032 4, 45884796063747006949801532473510 2,
45460722162973005037141629215899 2, 46436092134753209436259406708404 5,
45672759113360005993471580844704 7, 46181647794288808288663464753837 9,
46224055184366208479929455079599O, 45545536943127805419673609867421 4,
45672759113360005993471580844704 7, 46012018233979207523599503450793 5,
46012018233979207523599503450793 5, 46054425624056607714865493776554 6,
45587944333205205610939600193182 5, 46054425624056607714865493776554 6,
45545536943127805419673609867421 4, 46054425624056607714865493776554 6,
46181647794288808288663464753837 9, 46096833014134007906131484102315 7,
46139240404211408097397474428076 8, 45799981283592206567269551821988 0,
46775351255372410966387329314493 3, 46224055184366208479929455079599O,
45799981283592206567269551821988 0, 46775351255372410966387329314493 3,
46224055184366208479929455079599O, 46436092134753209436259406708404 5,
45884796063747006949801532473510 2, 46181647794288808288663464753837 9,
46054425624056607714865493776554 6, 45587944333205205610939600193182 5,
45927203453824407141067522799271 3, 45460722162973005037141629215899 2,
46181647794288808288663464753837 9, 45799981283592206567269551821988 0,
46224055184366208479929455079599O, 45672759113360005993471580844704 7,
46181647794288808288663464753837 9, 46648129085140210392589358337210 0,
45757573893514806376003561496226 9, 46096833014134007906131484102315 7,
46224055184366208479929455079599O, 46436092134753209436259406708404 5,
45799981283592206567269551821988 0, 46224055184366208479929455079599O,
45460722162973005037141629215899 2, 46224055184366208479929455079599O,
46648129085140210392589358337210 0, 46181647794288808288663464753837 9,
45927203453824407141067522799271 3, 45799981283592206567269551821988 0,
```

```
46224055184366208479929455079599O, 46012018233979207523599503450793S,
46605721695062810201323368011448S, 45842388673669606758535542147749I,
46224055184366208479929455079599O, 45672759113360005993471580844704I,
45884796063747006949801532473510Z, 46181647794288808288663464753837S,
46224055184366208479929455079599O, 45503129553050405228407619541660З,
45672759113360005993471580844704I, 45757573893514806376003561496226Ч,
45672759113360005993471580844704I, 45630351723282605802205590518943б,
45799981283592206567269551821988О, 46012018233979207523599503450793S,
46054425624056607714865493776554б, 45587944333205205610939600193182З,
46054425624056607714865493776554б, 46096833014134007906131484102315Z,
46139240404211408097397474428076Ч, 46054425624056607714865493776554б,
46096833014134007906131484102315Z, 46266462574443608671195445405360І,
46181647794288808288663464753837S, 45884796063747006949801532473510Z,
45460722162973005037141629215899Ю, 45757573893514806376003561496226Ч,
46096833014134007906131484102315Z, 46224055184366208479929455079599O,
46012018233979207523599503450793S, 45799981283592206567269551821988О,
45884796063747006949801532473510Z, 46224055184366208479929455079599O,
46054425624056607714865493776554б, 46012018233979207523599503450793S,
45672759113360005993471580844704I, 45503129553050405228407619541660З,
46266462574443608671195445405360І, 45630351723282605802205590518943б,
45672759113360005993471580844704I, 46224055184366208479929455079599O,
46224055184366208479929455079599O, 46436092134753209436259406708404З,
45884796063747006949801532473510Z, 46012018233979207523599503450793S,
46054425624056607714865493776554б, 45503129553050405228407619541660З,
46054425624056607714865493776554б, 45927203453824407141067522799271З,
46648129085140210392589358337210О, 46478499524830609627525397034165б,
45799981283592206567269551821988О, 46012018233979207523599503450793S,
46181647794288808288663464753837S, 46224055184366208479929455079599O,
45545536943127805419673609867421Ч, 46266462574443608671195445405360І,
46012018233979207523599503450793S, 46266462574443608671195445405360І,
45715166503437406184737571170465Ш, 46012018233979207523599503450793S,
45460722162973005037141629215899Ю, 46393684744675809244993416382643Ч,
45672759113360005993471580844704I, 46012018233979207523599503450793S,
45460722162973005037141629215899Ю, 45757573893514806376003561496226Ч,
45460722162973005037141629215899Ю, 46224055184366208479929455079599O,
46605721695062810201323368011448S, 46181647794288808288663464753837S,
46775351255372410966387329314493З, 46436092134753209436259406708404З,
46266462574443608671195445405360І, 45757573893514806376003561496226Ч,
46012018233979207523599503450793S, 45460722162973005037141629215899Ю,
45884796063747006949801532473510Z, 46054425624056607714865493776554б,
45545536943127805419673609867421Ч, 45503129553050405228407619541660З,
45672759113360005993471580844704I, 45757573893514806376003561496226Ч,
46096833014134007906131484102315Z, 46139240404211408097397474428076Ч,
46054425624056607714865493776554б, 45503129553050405228407619541660З,
45672759113360005993471580844704I, 45927203453824407141067522799271З,
46054425624056607714865493776554б, 45545536943127805419673609867421Ч,
46224055184366208479929455079599O, 46436092134753209436259406708404З,
45884796063747006949801532473510Z]

Расшифрованный текст:
вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходящ
ийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпубликац
ий.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.номож
```

```
ноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднукартин
у.текстнатысячусимволовэтосколькопримерноослов.статистикапоказывает,чтотысячавк
лючаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблятьпредл
огами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеизменнов
озрастает.вкопирайтерскойдеятельноститипринятосчитатьтысячиспробеламиилибез.учет
пробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименностолькоразмы
разделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,таккакэтоп
устоеместо.однаконекоторыефирмыибиржевидятсправедливымставитьстоимостьзатысячу
символовспробелами,считаяпоследниеважнымэлементомкачественноговосприятия.согла
ситесь,читатьслитныйтекстбезединогопропуска,никтонебудет.нобольшинствунужнацен
азатысячузнаковбезпробелов.
```

Блок-схема:



Блок I: АЛГОРИТМЫ ЦИФРОВЫХ ПОДПИСЕЙ

24.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
```

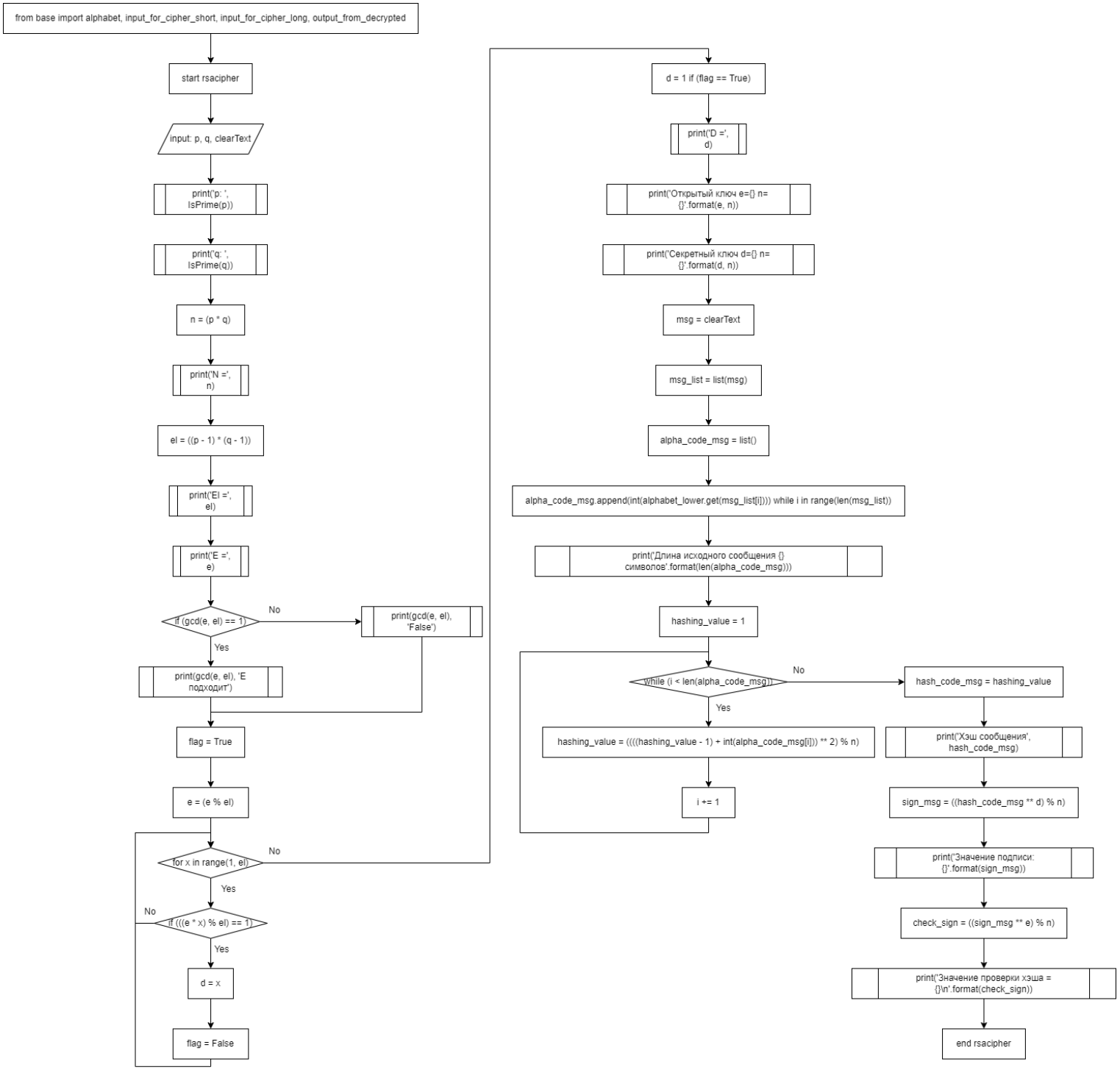
```
def rsacipher(p, q, clearText):
    p = int(p)
    print('p: ', IsPrime(p))
    q = int(q)
    print('q: ', IsPrime(q))
    n = p * q
    print("N =", n)
    el = (p-1) * (q-1)
    print("El =", el)
    e = 7
    print("E =", e)
    if gcd(e, el) == 1:
        print(gcd(e, el), "E подходит")
   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


from base import alphabet, input_for_cipher_short, input_for_cipher_long, output_from_decrypted
start rsacipher
input: p, q, clearText
print('p: ', IsPrime(p))
print('q: ', IsPrime(q))
n = (p * q)
print('N =', n)
el = ((p - 1) * (q - 1))
print('El =', el)
print('E =', e)
if (gcd(e, el) == 1)
No
print(gcd(e, el), 'False')
Yes
print(gcd(e, el), 'E подходит')
flag = True
e = (e % el)
for x in range(1, el)
No
Yes
if (((e * x) % el) == 1)
No
Yes
d = x
flag = False
d = 1 if (flag == True)
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()
alpha_code_msg.append(int(alphabet_lower.get(msg_list[i]))) while i in range(len(msg_list))
print('Длина исходного сообщения {} символов'.format(len(alpha_code_msg)))
hashing_value = 1
while (i < len(alpha_code_msg))
No
Yes
hashing_value = ((((hashing_value - 1) + int(alpha_code_msg[i])) ** 2) % n)
i += 1
hash_code_msg = hashing_value
print('Хэш сообщения', hash_code_msg)
sign_msg = ((hash_code_msg ** d) % n)
print('Значение подписи: {}'.format(sign_msg))
check_sign = ((sign_msg ** e) % n)
print('Значение проверки хэша = {}\n'.format(check_sign))
end rsacipher

25.El Gamal

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

Код программы:

```
from math import gcd
import random
from base import alphabet, input_for_cipher_short, input_for_cipher_long,
output_from_decrypted
alphavit = {'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
```

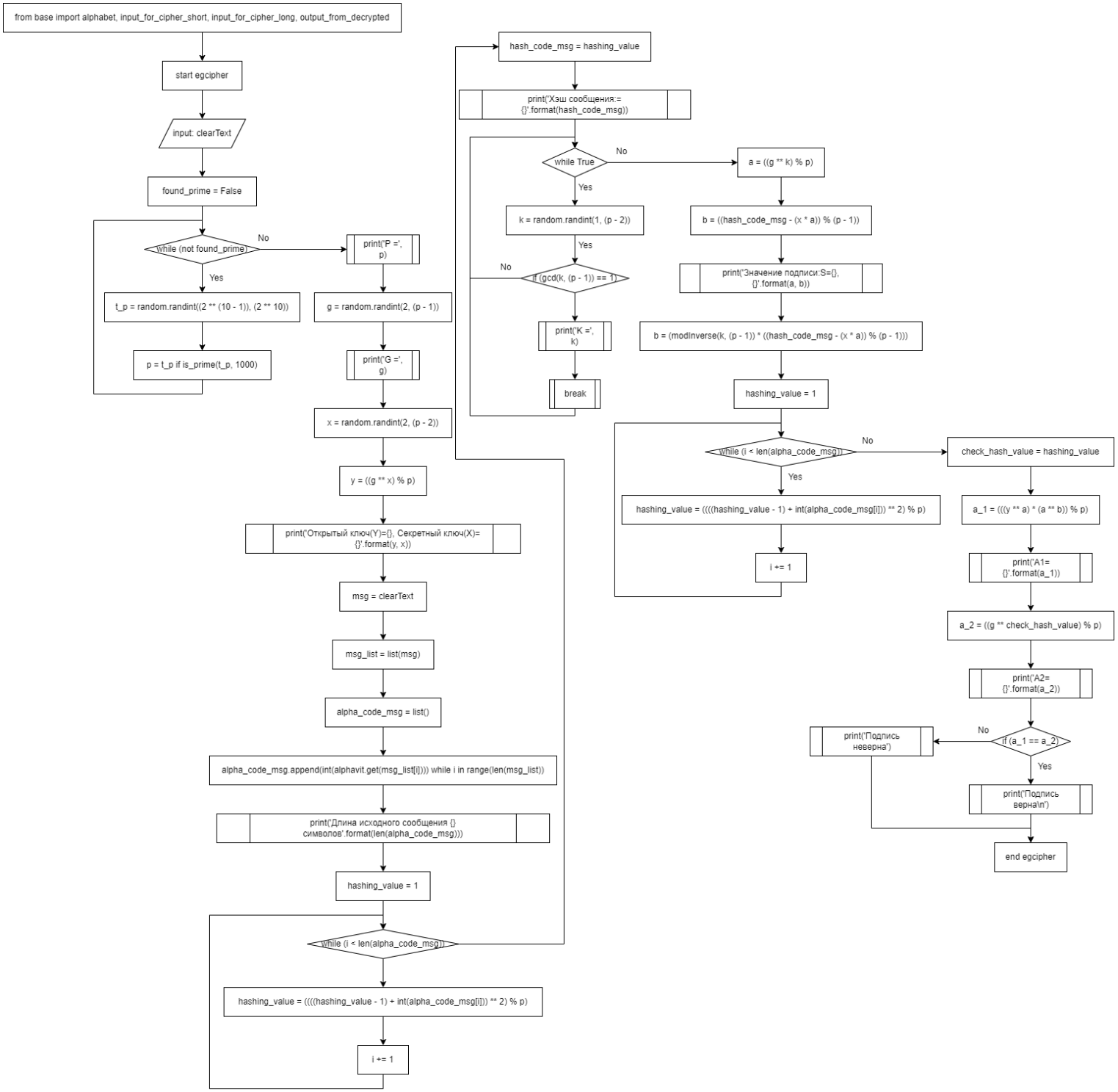
```
def hash_value(mod, alpha_code_msg):
    i = 0
    hashing_value = 1
    while i < len(alpha_code_msg):
        hashing_value = (((hashing_value-1) + int(alpha_code_msg[i]))**2) %
mod
        i += 1
    return hashing_value

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:


from base import alphabet, input_for_cipher_short, input_for_cipher_long, output_from_decrypted
start egcipher
input: clearText
found_prime = False
while (not found_prime)
No
Yes
t_p = random.randint((2 ** (10 - 1)), (2 ** 10))
p = t_p if is_prime(t_p, 1000)
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()
alpha_code_msg.append(int(alphavit.get(msg_list[i])) while i in range(len(msg_list))
print('Длина исходного сообщения {} символов'.format(len(alpha_code_msg)))
hashing_value = 1
while (i < len(alpha_code_msg))
hashing_value = ((((hashing_value - 1) + int(alpha_code_msg[i])) ** 2) % p)
i += 1
hash_code_msg = hashing_value
print('Хэш сообщения:= {}'.format(hash_code_msg))
while True
No
Yes
k = random.randint(1, (p - 2))
Yes
No
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)))
hashing_value = 1
while (i < len(alpha_code_msg))
No
Yes
hashing_value = ((((hashing_value - 1) + int(alpha_code_msg[i])) ** 2) % p)
i += 1
check_hash_value = hashing_value
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)
No
Yes
print('Подпись неверна')
print('Подпись верна\n')
end egcipher

Блок J: СТАНДАРТЫ ЦИФРОВЫХ ПОДПИСЕЙ

26.ГОСТ Р 34.10-94

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

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

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

x - некоторое число, меньшее q;

y = ax mod p.

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

Код программы:

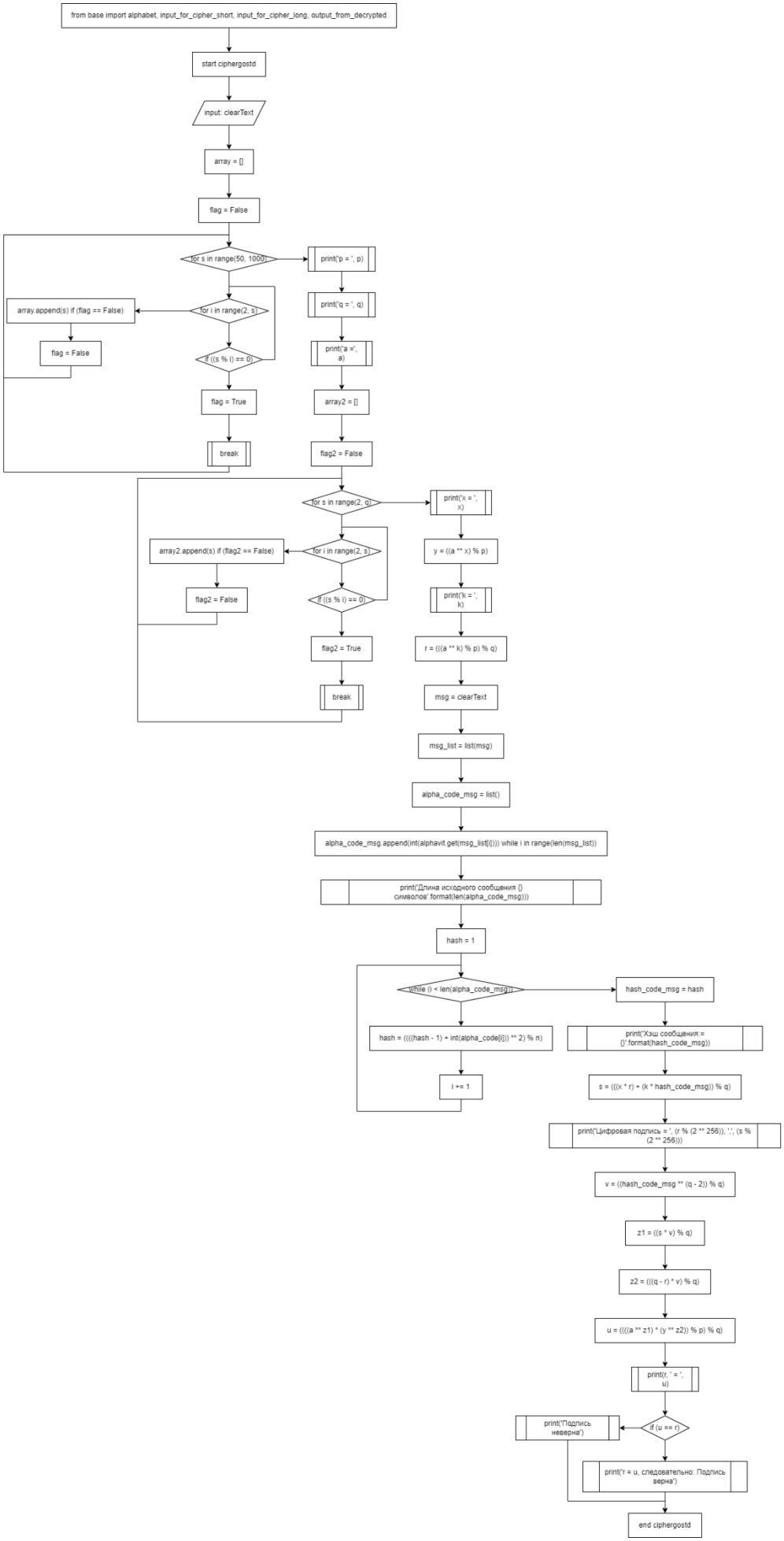
```
from base import alphabet, input_for_cipher_short, input_for_cipher_long,
output_from_decrypted

alphavit = {'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:


from base import alphabet, input_for_cipher_short, input_for_cipher_long, output_from_decrypted
start ciphergostd
input: clearText
array = []
flag = False
for s in range(50, 1000)
print('p = ', p)
print('q = ', q)
print('a =', a)
for i in range(2, s)
array.append(s) if (flag == False)
flag = False
if ((s % i) == 0)
flag = True
break
array2 = []
flag2 = False
for s in range(2, q)
print('x = ', x)
array2.append(s) if (flag2 == False)
for i in range(2, s)
flag2 = False
if ((s % i) == 0)
flag2 = True
break
y = ((a ** x) % p)
print('k = ', k)
r = (((a ** k) % p) % q)
msg = clearText
msg_list = list(msg)
alpha_code_msg = list()
alpha_code_msg.append(int(alphavit.get(msg_list[i]))) while i in range(len(msg_list))
print('Длина исходного сообщения {} символов'.format(len(alpha_code_msg)))
hash = 1
while (i < len(alpha_code_msg))
hash = ((((hash - 1) + int(alpha_code[i])) ** 2) % n)
i += 1
hash_code_msg = hash
print('Хэш сообщения = {}'.format(hash_code_msg))
s = (((k * 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('Подпись неверна')
print('r = u, следовательно: Подпись верна')
end ciphergostd

Блок K: Обмен ключами

28.ОБМЕН КЛЮЧАМИ ПО ДИФФИ-ХЕЛЛМАНУ

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

1. Определить секретные ключи пользователей KA и KB.

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 K: KA=YKA mod n KB=YKB mod n.

KA = KB = K

Код программы:

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

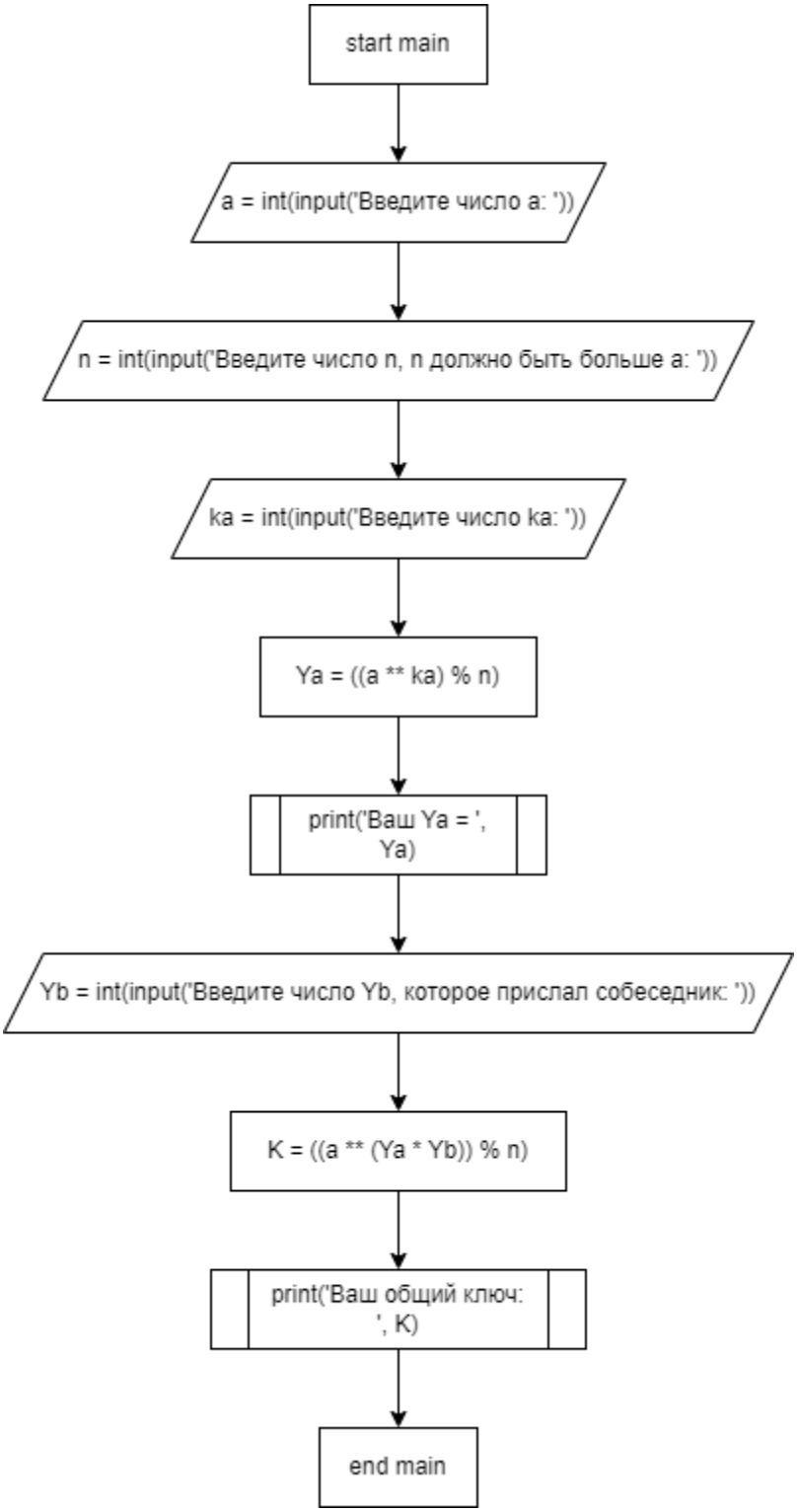
Тестирование:

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

Проверка:

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

Блок-схема:


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