

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



**МОСКОВСКИЙ ПОЛИТЕХ**

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ  
КАФЕДРА «ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ»

Отчёт

По дисциплине: «Программирование криптографических алгоритмов»

Группа: 191-351

Давыткина П.Е.

Проверила: Бутакова Н.Г.

Москва 2020 г.

## Оглавление

Аннотация.....	4
Блок А: ШИФРЫ ОДНОЗНАЧНОЙ ЗАМЕНЫ .....	11
1. Шифр АТБАШ .....	11
2. Шифр Цезаря.....	16
3. Квадрат Полибия.....	21
Блок В: ШИФРЫ МНОГОЗНАЧНОЙ ЗАМЕНЫ .....	26
4. Шифр Тритемия .....	26
5. Шифр Белазо .....	31
6. Шифр Виженера.....	36
Блок С: ШИФРЫ БЛОЧНОЙ ЗАМЕНЫ.....	42
8. Матричный шифр.....	42
Блок D: ШИФРЫ ПЕРЕСТАНОВКИ.....	49
10. Шифр Вертикальной Перестановки.....	49
Блок Е: ШИФРЫ ГАММИРОВАНИЯ .....	55
13. Одноразовый блокнот Шеннона.....	55
Блок F: ПОТОЧНЫЕ ШИФРЫ .....	60
15. A5 /1 .....	60
16. A5 /2 .....	68
Блок G: КОМБИНАЦИОННЫЕ ШИФРЫ .....	76
17.МАГМА .....	76
20.КУЗНЕЧИК.....	82
БЛОК Н: АСИММЕТРИЧНЫЕ ШИФРЫ .....	85
21. Шифр RSA .....	85

Блок I: АЛГОРИТМЫ ЦИФРОВЫХ ПОДПИСЕЙ.....	91
24. Шифр RSA .....	91
Блок J: СТАНДАРТЫ ЦИФРОВЫХ ПОДПИСЕЙ .....	98
28.ГОСТ Р 34.10-2012 .....	98
БЛОК К: ОБМЕН КЛЮЧАМИ .....	107
29. Обмен ключами по Diffi-Hellman.....	107

## **Аннотация**

**Среда программирования:** Visual Studio Code (VS Code) — редактор исходного кода, разработанный Microsoft для Windows, Linux и macOS. Позиционируется как «лёгкий» редактор кода для кроссплатформенной разработки веб- и облачных приложений.

**Язык программирования:** Python — высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью. Язык является полностью объектно-ориентированным — всё является объектами.

**Процедуры для запуска программы:** Нажать на кнопку «Run Python File»

**Пословица-текст:** Леопард не может изменить своих пятен.

**Текст для проверки работы (1012 знаков):** Людмила Петрушевская. Будильник. Жил, да был будильник. У него были усы, шляпа и сердце. И он решил жениться. Он решил жениться, когда стукнет без пятнадцати девять. Ровно в восемь он сделал предложение графину с водой. Графин с водой согласился немедленно, но в пятнадцать минут девятого его унесли и выдали замуж за водопроводный кран. Дело было сделано, и графин вернулся на стол к будильнику уже замужней дамой. Было двадцать минут девятого. Времени оставалось мало. Будильник тогда сделал предложение очкам. Очки были старые и неоднократно выходили замуж за уши. Очки подумали пять минут и согласились, но в этот момент их опять выдали замуж за уши. Было уже восемь часов двадцать пять минут. Тогда будильник быстро сделал предложение книге. Книга тут же согласилась, и будильник стал ждать, когда же стукнет без пятнадцати девять. Сердце его очень громко колотилось. Тут его взяли и накрыли

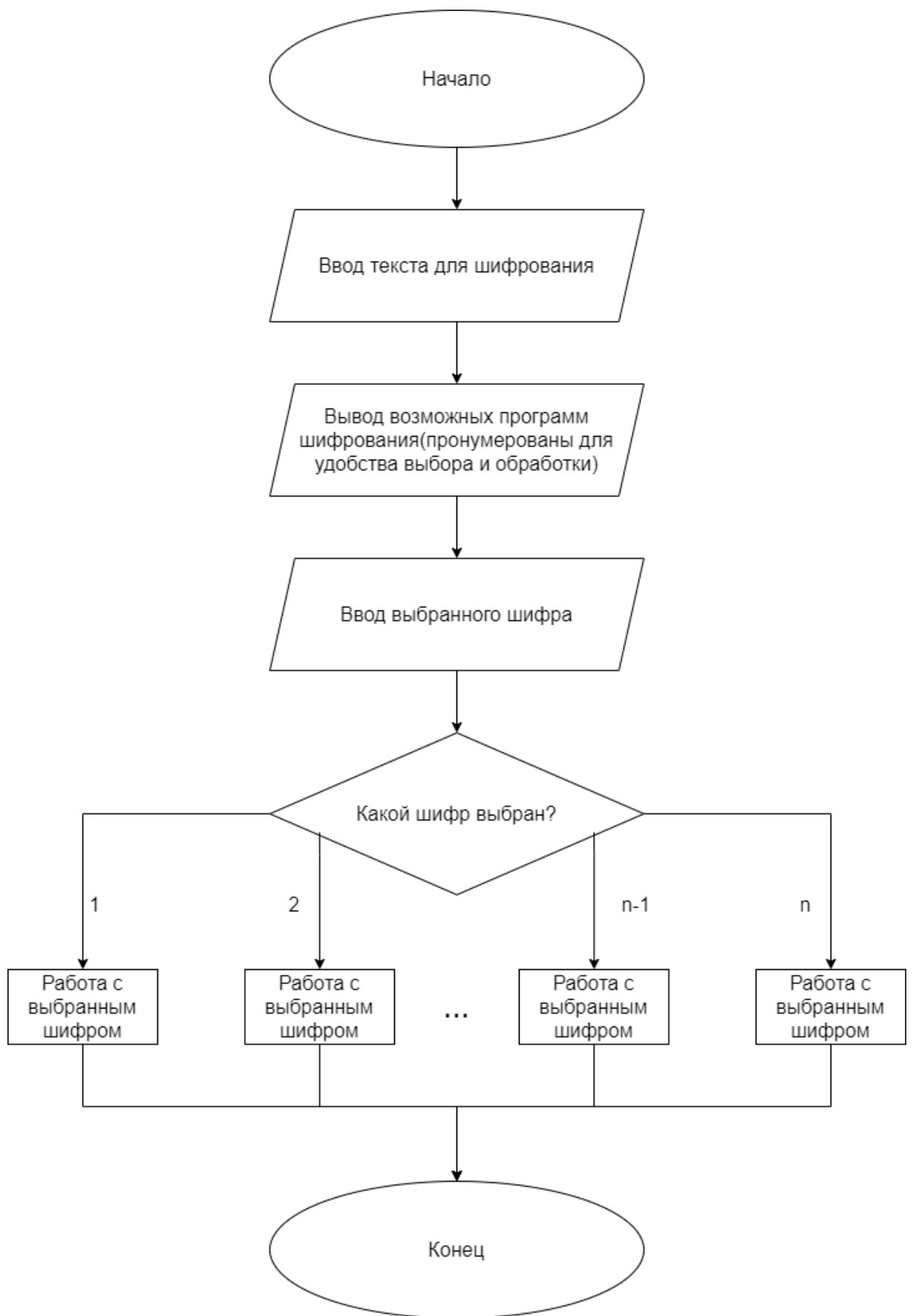
подушкой, потому что детей уложили спать. И без пятнадцати девять будильник неожиданно для себя женился на подушке.

### Интерфейс

#### Описание

Обычный консольный интерфейс, при запуске программы запрашивается выбор шифра, далее введенный текст шифруется по выбранному алгоритму.

#### Блок-схема



Код программы-интерфейса

# ТЕЛО ПРОГРАММЫ

```

proverb = input("Введите текст:\n")
# proverb = "Леопард не может изменить своих пятен."
# proverb = "абвгдежзийклмнопрстуфхцщъыьэюя"

print ("Выберете шифр, которым хотели бы зашифровать: \n ШИФРЫ ОДНОЗНАЧНОЙ ЗАМЕНЫ \n 1 -
АТБАШ \n 2 - Шифр Цезаря \n 3 - Квадрат Полибия")
print("ШИФРЫ МНОГОЗНАЧНОЙ ЗАМЕНЫ \n 4 - Шифр Белазо \n 5 - Шифр Тритемия \n 6 - Шифр
Виженера ")
print("ШИФРЫ БЛОЧНОЙ ЗАМЕНЫ \n 7 - Шифр Плейфера \n 8 - Матричный шифр ")
print("ШИФР ПЕРСТАНОВКИ \n 9 - Шифр Вертикальной Перестановки \n ШИФР ГАММИРОВАНИЯ \n 10-
Одноразовый блокнот Шеннона \n ПОТОЧНЫЙ ШИФР \n 11 - A5/1 ")
print(" КОМБИНАЦИОННЫЙ ШИФР \n 12 - Кузнечик \n АССИМЕТРИЧНЫЕ ШИФРЫ(ГЕНЕРАЦИЯ ЦИФРОВОЙ
ПОДПИСИ) \n 13 - RSA \n 14 - Elgamal \n 15 - Обмен ключами по алгоритму Diffie-Hellman")
code_cp = input()

# АТБАШ
if code_cp == "1": # проверяем, какой шифр вызвал пользователь
    proverb = change_line(proverb) # вызываем функцию, которая преобразует строку
    cryp_text = cryp_atb(proverb) # вызываем функцию шифрования
    print("Зашифровано(АТБАШ)")
    print(cryp_text)
    decryp_text = dec_atb(cryp_text) # вызываем функцию расшифровки
    print("Расшифрованно(АТБАШ)")
    print(decryp_text)

# Цезарь
elif code_cp == "2": # проверяем, какой шифр вызвал пользователь
    proverb = change_line(proverb) # вызываем функцию, которая преобразует строку
    print("Введите ключ для шифрования")
    key = input()
    key = int(key) #преобразуем введенный ключ в целочисленный формат
    cryp_text = cryp_c(proverb, key) # вызываем функцию шифрования
    print("Зашифровано(Цезарь)")
    print(cryp_text)
    print("Введите ключ для расшифровки")
    key = input()
    key = int(key) #преобразуем введенный ключ в целочисленный формат
    decryp_text = dec_c(cryp_text, key) # вызываем функцию расшифровки
    print("Расшифрованно(Цезарь)")
    print(decryp_text)

# Белазо
elif code_cp == "4": # проверяем, какой шифр вызвал пользователь
    proverb = change_line(proverb) # вызываем функцию, которая преобразует строку
    print("Введите ключ для шифрования")
    key = input() # вводим ключ для шифрования
    cryp_text = cryp_bel(proverb, key) # вызываем функцию шифрования
    print("Зашифровано(Белазо)")
    print(cryp_text)
    print("Введите ключ для расшифровки")
    key = input() # вводим ключ для расшифровки
    decryp_text = dec_bel(cryp_text, key) # вызываем функцию расшифровки
    print("Расшифрованно(Белазо)")
    print(decryp_text)

# Квадрат Полибия
elif code_cp == "3": # проверяем, какой шифр вызвал пользователь
    proverb = change_line(proverb) # вызываем функцию, которая преобразует строку

```

```

cryp_text = cryp_qp(proverb) # вызываем функцию шифрования
print("Зашифровано(Квадрат Полибия)")
print(cryp_text)
decryp_text = dec_qp(cryp_text)
print("Расшифровано(Квадрат Полибия)") # вызываем функцию расшифровки
print(decryp_text)

# Шифр Тритемия
elif code_cp == "5": # проверяем, какой шифр вызвал пользователь
    proverb = change_line(proverb) # вызываем функцию, которая преобразует строку

    cryp_text = cryp_try(proverb) # вызываем функцию шифрования
    print("Зашифровано(Шифр Тритемия)")
    print(cryp_text)

    decryp_text = dec_try(cryp_text) # вызываем функцию расшифровки
    print("Расшифровано(Шифр Тритемия)")
    print(decryp_text)

# Шифр Виженера
elif code_cp == "6": # проверяем, какой шифр вызвал пользователь
    proverb = change_line(proverb) # вызываем функцию, которая преобразует строку
    n = len(proverb)
    print("Введите ключ для шифрования")
    key = input() # вводим ключ для шифрования
    key += proverb[:n-1]
    cryp_text = cryp_vig(proverb, key) # вызываем функцию шифрования
    print("Зашифровано(Шифр Виженера)")
    print(cryp_text)
    n = len(proverb)
    print("Введите ключ для шифрования")
    key = input() # вводим ключ для расшифровки
    key += proverb[:n-1]
    decryp_text = dec_vig(cryp_text, key) # вызываем функцию расшифровки
    print("Расшифровано(Шифр Виженера)")
    print(decryp_text)

# Шифр Вертикальной Перестановки
elif code_cp == "9": # проверяем, какой шифр вызвал пользователь
    proverb = change_line(proverb) # вызываем функцию, которая преобразует строку
    print("Введите ключ для шифрования")
    key = input() # вводим ключ для шифрования
    cryp_text = cryp_ver(proverb, key) # вызываем функцию шифрования
    print("Зашифровано(Шифр Вертикальной Перестановки)")
    cryp = cryp_text.replace("-", "")
    print(cryp)
    print("Введите ключ для расшифровки")
    key = input() # вводим ключ для расшифровки
    decryp_text = dec_ver(cryp_text, key) # вызываем функцию расшифровки
    print("Расшифровано(Шифр Вертикальной Перестановки)")
    print(decryp_text)

# Одноразовый блокнот Шеннона
elif code_cp == "10": # проверяем, какой шифр вызвал пользователь
    proverb = change_line(proverb) # вызываем функцию, которая преобразует строку

    print("Генерируем ключ для шифрования")
    key=[random.randint(1,32) for i in range(len(proverb))] #генеринуем ключ
    print(key)
    cryp_text = cryp_sh(proverb, key) # вызываем функцию шифрования

```



```

print("Зашифровано(Одноразовый блокнот Шеннона)")
print(cryp_text)
decryp_text = dec_sh(cryp_text, key) # вызываем функцию расшифровки
print("Расшифровано(Одноразовый блокнот Шеннона)")
print(decryp_text)

# RSA
elif code_cp == "13": # проверяем, какой шифр вызвал пользователь
    proverb = change_line(proverb) # вызываем функцию, которая преобразует строку

    p = int(input("Введите значение параметра P(простое число:"))
    p = prime_num(p) # проверяем параметр p на простоту
    q = int(input("Введите значение параметра Q(простое число:"))
    q = prime_num(q) # проверяем параметр q на простоту
    n=p*q
    f = (p-1)*(q-1)
    print("Введите значение параметра E, взаимно простое", f, ": ")
    e = int(input())
    e = int(is_coprime(e, f)) # проверяем является ли e взаимно простое f

    cryp_text = cryp_rsa(proverb, n, e) # вызываем функцию шифрования
    print("Зашифровано(RSA)")
    print(cryp_text)
    decryp_text = dec_rsa(cryp_text, n, e, f) # вызываем функцию расшифровки
    print("Расшифровано(RSA)")
    print(decryp_text)

# Elgamal
elif code_cp == "14": # проверяем, какой шифр вызвал пользователь
    proverb = change_line(proverb) # вызываем функцию, которая преобразует строку
    p = int(input("Введите p > 32:"))
    g = random.randint(1,p)
    x = random.randint(1,p)
    y = (g ** x) % p
    print("Открытые ключи p =", p, " g =", g, " y =", y)
    f = p - 1

    cryp_text = cryp_elm(proverb,p,g,y,f)
    print("Зашифровано(Elgamal)")
    print(cryp_text)
    decryp_text = dec_elm(cryp_text)
    print("Расшифровано(Elgamal)")
    print(decryp_text)

# Обмен ключами по алгоритму Diffie-Hellman
elif code_cp == "15": # проверяем, какой шифр вызвал пользователь
    print("Введите a и n , удовлетворяющие условию 1 < a < n \n Введите a")
    a = int(input())
    print("Введите n")
    n = int(input())

    if (a > 1) and (a < n):
        exchange_key(n, a)
    else:
        print("Некорректно введены данные")

```

```
else:  
    print("Ошибка проверите корректность введенных данных")
```

## Блок А: ШИФРЫ ОДНОЗНАЧНОЙ ЗАМЕНЫ

### 1. Шифр АТБАШ

Алгоритм шифра, описание:

$$Y_i = X_{(n-i+1)}$$

**X** – исходный (открытый) текст

**Y**– зашифрованный текст

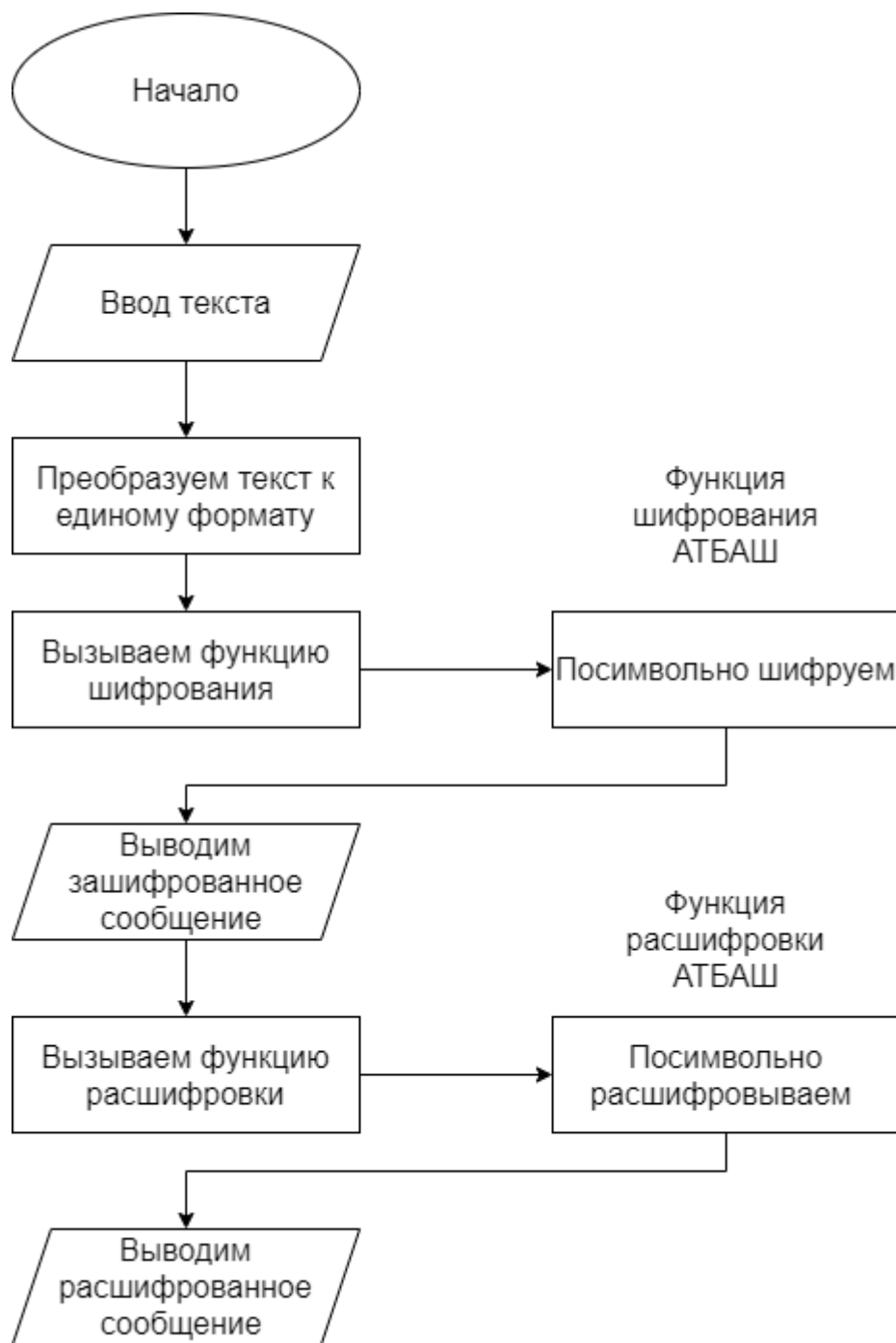
**i** – порядковый номер буквы в открытом алфавите,  $i=1 \dots n$

**n** – количество букв в открытом алфавите.

У	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
Х	Я	Ю	Э	Ь	Ы	Ъ	Щ	Ш	Ч	Ц	Х	Ф	У	Т	С	Р
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

У	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
Х	П	О	Н	М	Л	К	Й	И	З	Ж	Е	Д	Г	В	Б	А
i	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

Блок-схема программы:



Код программы с комментариями:

```

# ШИФР АТБАШ
def cryp_atb(pr): #функция шифрования
    i = 0
    cryp_text = ""
    n = len(pr)
    for i in range(n):
        symb = alph[31 - alph.find(pr[i])] # высчитываем символ
        cryp_text = cryp_text + symb
    return cryp_text

def dec_atb(pr): #функция расшифровки
    pr = pr.replace(" ", "") # убираем пробелы в тексте, который нужно расшифровать
    pr.lower() #делаем все буквы маленькими
  
```

```

decryp_text = ""
n = len(pr)
i = 0
for i in range(n):
    symb = d_alph[31 - d_alph.find(pr[i])] # высчитываем символ
    decryp_text = decryp_text + symb
return (decryp_text)

# КОНЕЦ ШИФР АТБАШ

Часть кода с вызовом функций:

if code_cp == "1": # проверяем, какой шифр вызвал пользователь
    proverb = change_line(proverb) # вызываем функцию, которая преобразует строку
    cryp_text = cryp_atb(proverb) # вызываем функцию шифрования
    print("Зашифровано(АТБАШ)")
    print(cryp_text)
    decryp_text = dec_atb(cryp_text) # вызываем функцию расшифровки
    print("Расшифровано(АТБАШ)")
    print(decryp_text)

```

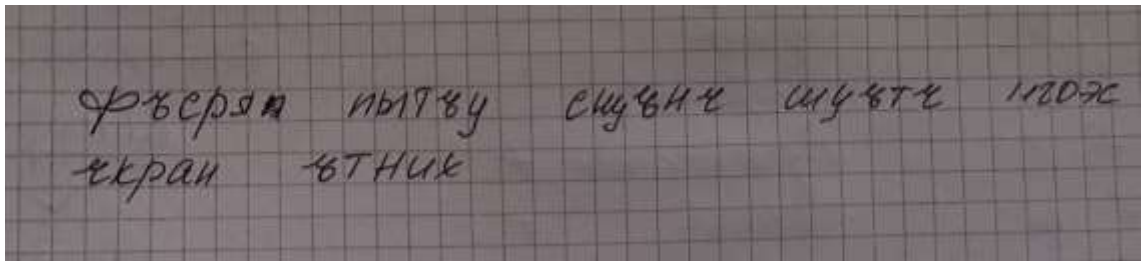
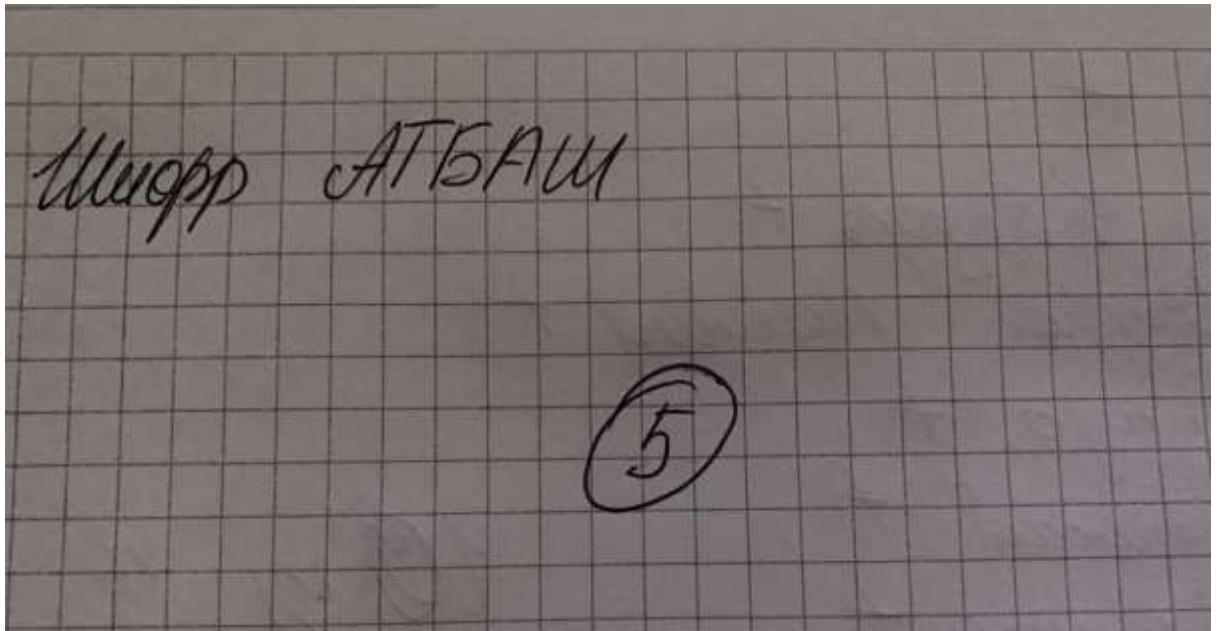
## Тестирование:

```

Выберете шифр, которым хотели бы зашифровать:
ШИФРЫ ОДНОЗНАЧНОЙ ЗАМЕНЫ
1 - АТБАШ
2 - Шифр Цезаря
3 - Квадрат Полибия
ШИФРЫ МНОГОЗНАЧНОЙ ЗАМЕНЫ
4 - Шифр Белазо
5 - Шифр Тритемия
6 - Шифр Виженера
ШИФРЫ БЛОЧНОЙ ЗАМЕНЫ
7 - Шифр Плейфера
8 - Матричный шифр
ШИФР ПЕРСТАНОВКИ
9 - Шифр Вертикальной Перестановки
ШИФР ГАММИРОВАНИЯ
10- Одноразовый блокнот Шеннона
ПОТОЧНЫЙ ШИФР
11 - A5/1
КОМБИНАЦИОННЫЙ ШИФР
12 - Кузнечик
АССИМЕТРИЧНЫЕ ШИФРЫ(ГЕНЕРАЦИЯ ЦИФРОВОЙ ПОДПИСИ)
13 - RSA
14 - Elgamal
15 - Обмен ключами по алгоритму Diffie-Hellman
1
Зашифровано(АТБАШ)
фъсряпытъущънчуътчнгоэсчкранътних
Расшифровано(АТБАШ)
леопарднеможетизменитьсвоихпятентчк

```

## Карточки:



Работа с текстом не менее 1000 знаков:

Введите текст)

Будильник. Будильник. Вот, да был будильник. У него были уши, зало и сердце. И он решил мечтаться. Он решил мечтаться, когда стукнет без пятнадцати девять. Ровно в восемь он сделал предложение графе ну с водой. График с водой согласился немедленно, но в пятнадцать минут девятого его унесло и вышло. Мечу за водопроводный кран. Дело было сделано, и график вернулся на стол к будильнику уже запертой дверью. Было девять минут девятого. Будильник оставался один. Будильник тогда сделал предложение рыбе. Рыба была старой и неоднократно вышвырнула крана за уша. Она پذیرفته 999 минут и согласилась, но в этот момент на столе надлежало крана за уша. Было уже восемь часов двадцать пять минут. Тогда будильник быстро сделал предложение рыбе. Рыба тут же согласилась, и будильник стал ждать, когда же стукнет без пятнадцати девять. Сейчас уже очень поздно спохватываться. Тут кто-то вошел и накрыл будильник, потому что датчик указывал спать. И без пятнадцати девять будильник немедленно для себя начал на подорож.

Выберите шифр, которым хотите вы зашифровать)

Шифры однобуквенной азбуки)

1 - АТБАШ

2 - Шифр Цезаря

3 - Шифр Вижения

Шифры многобуквенной азбуки)

4 - Шифр Каспара

5 - Шифр Триграмм

6 - Шифр Бленкина

Шифры блочной азбуки)

7 - Шифр Полифена

8 - Полиграфический шифр

9 - Шифр Фридриха Бернхардта

Шифр симметричный

10 - Простой шифр Вернама

Полный шифр

11 - А5/1

Криптографический шифр

12 - Булевы

Алгоритмы шифрования (генерация двоичной последовательности)

13 - RSA

14 - ElGamal

15 - Обмен ключами по алгоритму Диффи-Хеллмана

1

Зашифровано (АТБАШ)

Ферма пытае екувнх шувтх изох экран втних

Будильник. Будильник. Вот, да был будильник. У него были уши, зало и сердце. И он решил мечтаться. Он решил мечтаться, когда стукнет без пятнадцати девять. Ровно в восемь он сделал предложение графе ну с водой. График с водой согласился немедленно, но в пятнадцать минут девятого его унесло и вышло. Мечу за водопроводный кран. Дело было сделано, и график вернулся на стол к будильнику уже запертой дверью. Было девять минут девятого. Будильник оставался один. Будильник тогда сделал предложение рыбе. Рыба была старой и неоднократно вышвырнула крана за уша. Она پذیرفته 999 минут и согласилась, но в этот момент на столе надлежало крана за уша. Было уже восемь часов двадцать пять минут. Тогда будильник быстро сделал предложение рыбе. Рыба тут же согласилась, и будильник стал ждать, когда же стукнет без пятнадцати девять. Сейчас уже очень поздно спохватываться. Тут кто-то вошел и накрыл будильник, потому что датчик указывал спать. И без пятнадцати девять будильник немедленно для себя начал на подорож.



## Блок А: ШИФРЫ ОДНОЗНАЧНОЙ ЗАМЕНЫ

### 2. Шифр Цезаря

Алгоритм шифра, описание:

$$Y_i = X_{i+3} \bmod n$$

**X** – исходный (открытый) текст

**Y** – зашифрованный текст

**i** – порядковый номер буквы открытого текста в алфавите,  $i=(1 \dots n)$

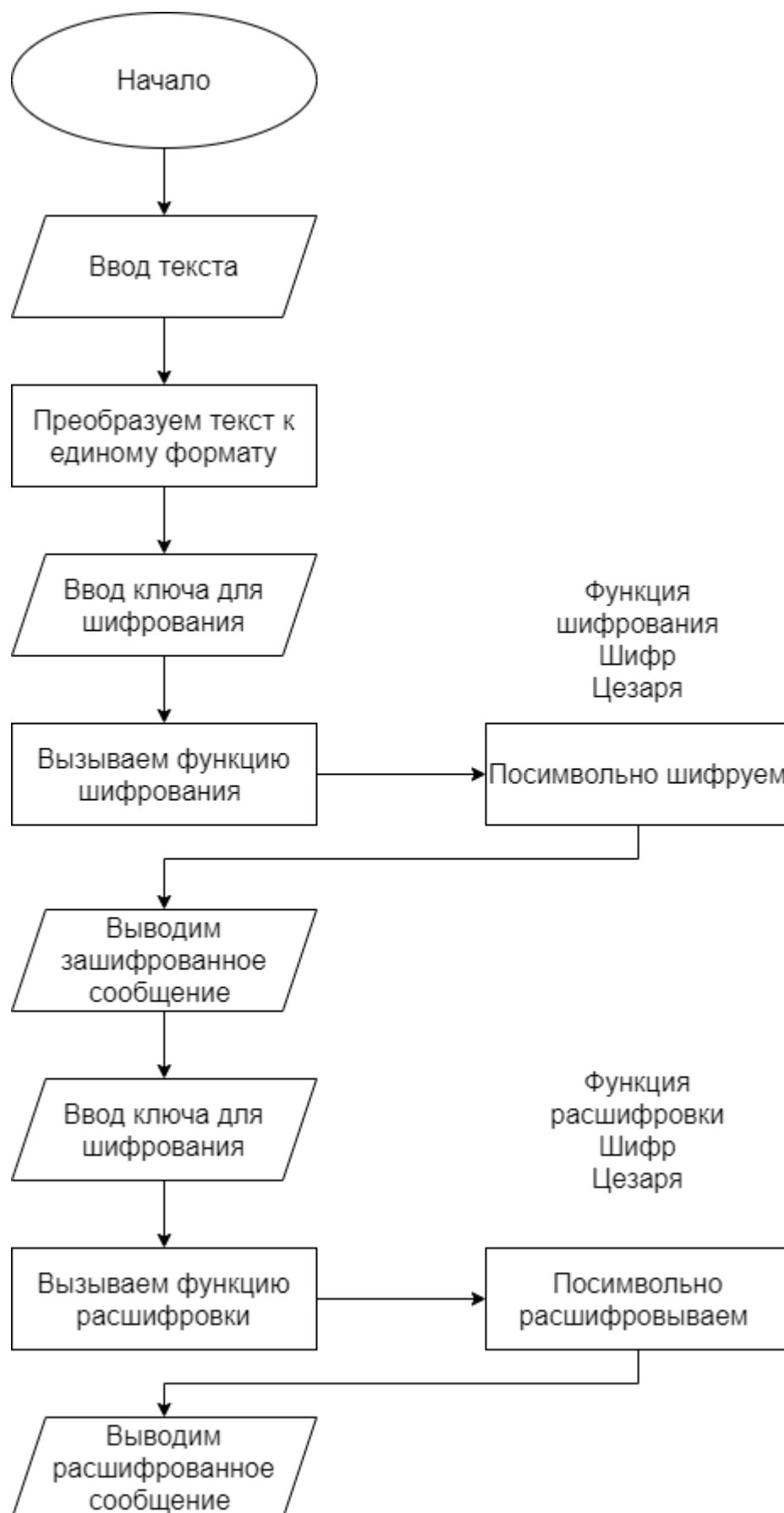
**n** – количество букв в выбранном алфавите (мощность алфавита).

<b>X</b>	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
<b>Y</b>	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т
<b>i</b>	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

<b>X</b>	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
<b>Y</b>	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	А	Б	В
<b>i</b>	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

Блок-схема программы:





## Код программы с комментариями:

```
# ШИФР ЦЕЗАРЯ
def cryp_c(pr, k): #функция шифрования
    i = 0
    cryp_text = ""
    n = len(pr)
    for i in range(n):
        symb = alph[(alph.find(pr[i]) + k) % 32] # ищем нужный символ
        cryp_text = cryp_text + symb
    return cryp_text

def dec_c(pr,k): #функция расшифровки
    pr = pr.replace(" ", "")
    pr = pr.lower()
    i=0
    n = len(pr)
    decryp_text = ""
    for i in range(n):
        symb = alph[(alph.find(pr[i]) - k + 32) % 32] #ищем нужный символ
        # print(symb)
        decryp_text = decryp_text + symb

    return decryp_text

# КОНЕЦ ШИФР ЦЕЗАРЯ
```

## Часть кода с вызовом функций:

```
# Цезарь
elif code_cp == "2": # проверяем, какой шифр вызвал пользователь
    proverb = change_line(proverb) # вызываем функцию, которая преобразует строку
    print("Введите ключ для шифрования")
    key = input()
    key = int(key) #преобразуем введенный ключ в целочисленный формат
    cryp_text = cryp_c(proverb, key) # вызываем функцию шифрования
    print("Зашифровано(Цезарь)")
    print(cryp_text)
    print("Введите ключ для расшифровки")
    key = input()
    key = int(key) #преобразуем введенный ключ в целочисленный формат
    decryp_text = dec_c(cryp_text, key) # вызываем функцию расшифровки
    print("Расшифровано(Цезарь)")
    print(decryp_text)
```

## Тестирование:

Выберете шифр, которым хотели бы зашифровать:

ШИФРЫ ОДНОЗНАЧНОЙ ЗАМЕНЫ

1 - АТБАШ

2 - Шифр Цезаря

3 - Квадрат Полибия

ШИФРЫ МНОГОЗНАЧНОЙ ЗАМЕНЫ

4 - Шифр Белазо

5 - Шифр Тритемия

6 - Шифр Виженера

ШИФРЫ БЛОЧНОЙ ЗАМЕНЫ

7 - Шифр Плейфера

8 - Матричный шифр

ШИФР ПЕРСТАНОВКИ

9 - Шифр Вертикальной Перестановки

ШИФР ГАММИРОВАНИЯ

10- Одноразовый блокнот Шеннона

ПОТОЧНЫЙ ШИФР

11 - A5/1

КОМБИНАЦИОННЫЙ ШИФР

12 - Кузнечик

АССИМЕТРИЧНЫЕ ШИФРЫ(ГЕНЕРАЦИЯ ЦИФРОВОЙ ПОДПИСИ)

13 - RSA

14 - Elgamal

15 - Обмен ключами по алгоритму Diffie-Hellman

2

Введите ключ для шифрования

3

Зашифровано(Цезарь)

оистгузрипсийхлкпирлхяфеслштвхирхън

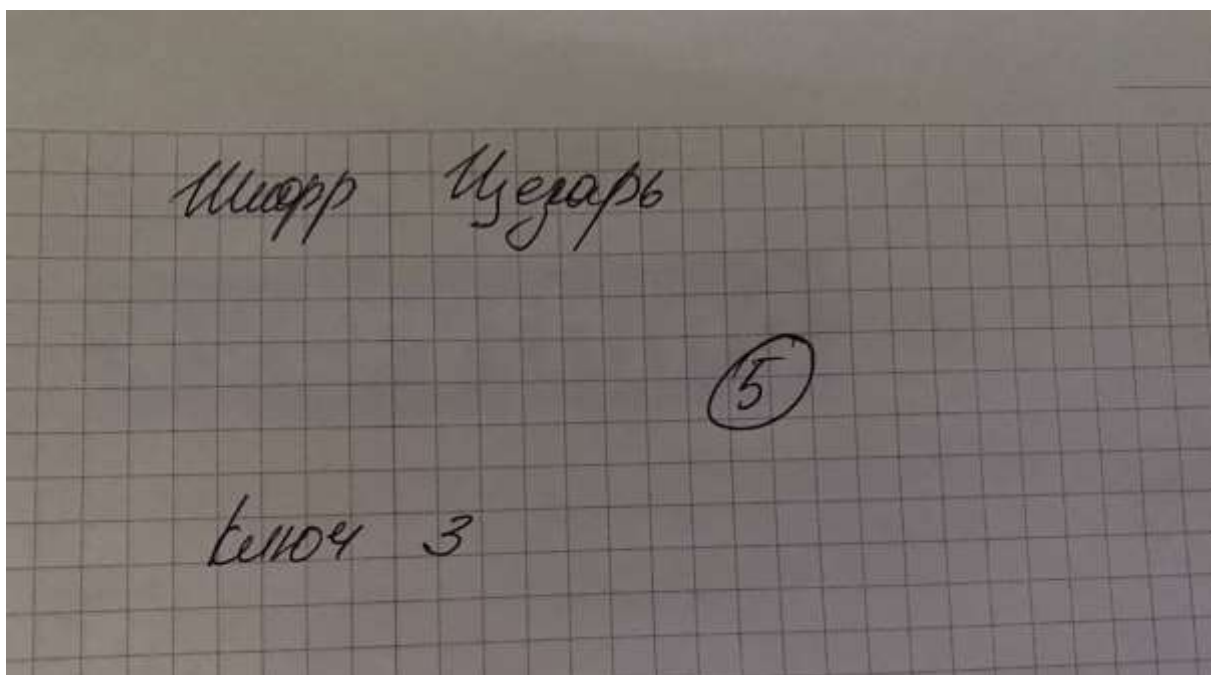
Введите ключ для расшифровки

3

Расшифрованно(Цезарь)

леопарднеможетизменитьсясвоихпятентчк

Карточки:





## Блок А: ШИФРЫ ОДНОЗНАЧНОЙ ЗАМЕНЫ

### 3. Квадрат Полибия

Алгоритм шифра, описание:

Шифрование производится по формуле:

$$Y_{ij} = i j$$

Где:

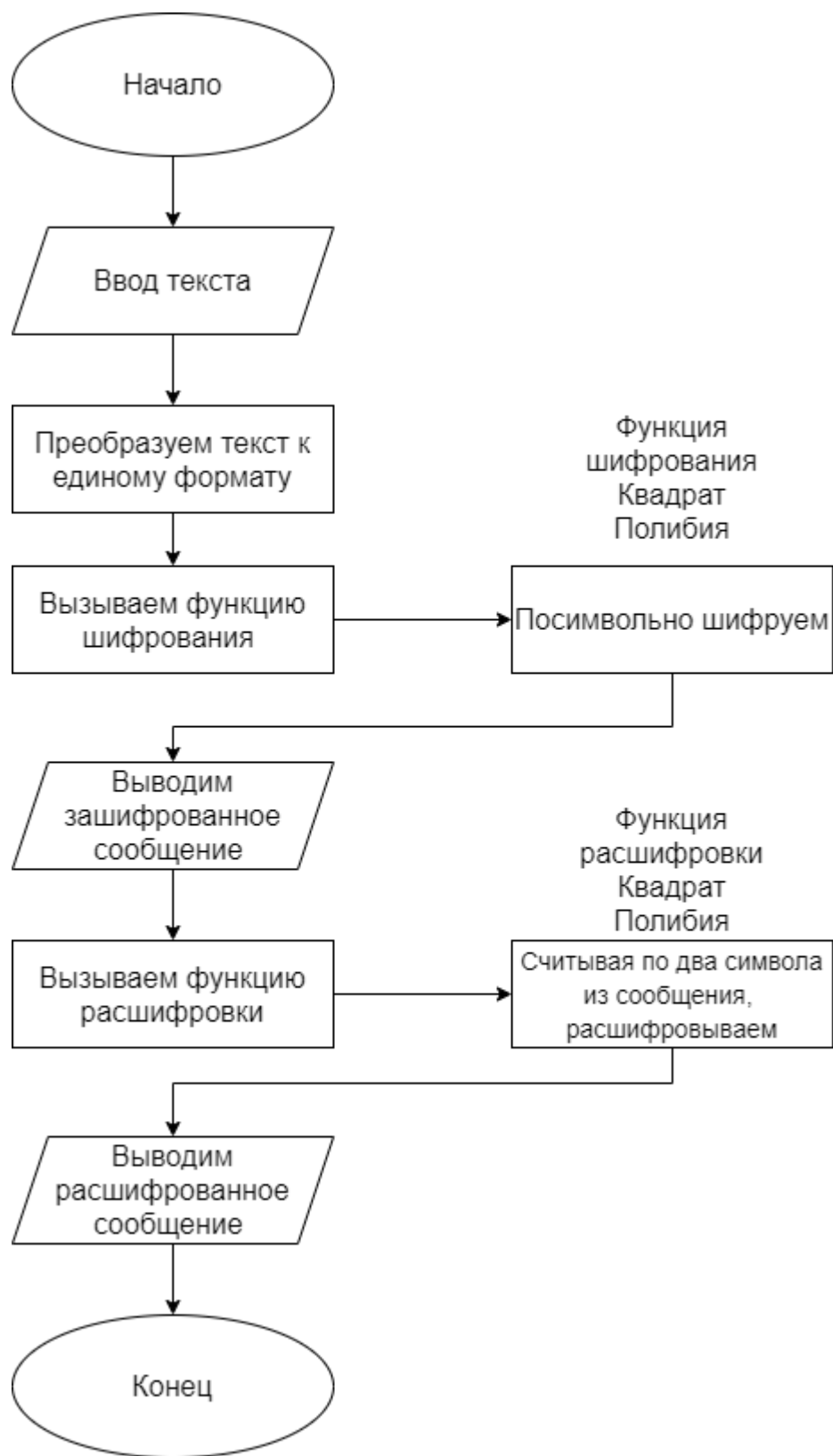
$Y$  – исходный (открытый) текст

$ij$  – зашифрованный текст

$i$  – номер строки

$j$  – номер столбца

Блок-схема программы:



Код программы с комментариями:

```
# КВАДРАТ ПОЛИБИЯ  
  
def crypt_qp(pr): # функция шифрования  
    i = 0
```

```

matr= [ ["a","б","в","г","д","е"], #матрица для удобства шифрования
        ["ё","ж","з","и","й","к"],
        ["л","м","н","о","п","р"],
        ["с","т","у","ф","х","ц"],
        ["ч","ш","щ","ъ","ы","ь"],
        ["э","ю","я", "-", "-","-"] ]

cryp_text = ""
n = len(pr)
for i in range(n):
    for j in range(6):
        for k in range(6):
            if pr[i] == matr[j][k]: # ищем букву в матрице
                cryp_text = cryp_text + str(j+1)+ str(k+1) # индексы буквы записываем

return cryp_text

def dec_qp(pr): # функция расшифровки
    pr = pr.replace(" ", "")

    i=0

    matr= [ ["a","б","в","г","д","е"], #матрица для удобства расшифровки
            ["ё","ж","з","и","й","к"],
            ["л","м","н","о","п","р"],
            ["с","т","у","ф","х","ц"],
            ["ч","ш","щ","ъ","ы","ь"],
            ["э","ю","я", "-", "-","-"] ]

    decryp_text = ""

    n = len(pr)
    for i in range(0, n - 1, 2): # читаем строку с шагом 2
        j = int(pr[i])
        k = int(pr[i+1])
        decryp_text = decryp_text + matr [j-1][k-1] # по найденным индексам записываем букву

    return decryp_text
# КОНЕЦ КВАДРАТ ПОЛИБИЯ

```

### Часть кода с вызовом функций:

```

elif code_cp == "3": # проверяем, какой шифр вызвал пользователь
    proverb = change_line(proverb) # вызываем функцию, которая преобразует строку
    cryp_text = cryp_qp(proverb) # вызываем функцию шифрования
    print("Зашифровано(Квадрат Полибия)")
    print(cryp_text)
    decryp_text = dec_qp(cryp_text)
    print("Расшифровано(Квадрат Полибия)") # вызываем функцию расшифровки
    print(decryp_text)

```

### Тестирование:

Выберете шифр, которым хотели бы зашифровать:

ШИФРЫ ОДНОЗНАЧНОЙ ЗАМЕНЫ

1 - АТБАШ

2 - Шифр Цезаря

3 - Квадрат Полибия

ШИФРЫ МНОГОЗНАЧНОЙ ЗАМЕНЫ

4 - Шифр Белазо

5 - Шифр Тритемия

6 - Шифр Виженера

ШИФРЫ БЛОЧНОЙ ЗАМЕНЫ

7 - Шифр Плейфера

8 - Матричный шифр

ШИФР ПЕРСТАНОВКИ

9 - Шифр Вертикальной Перестановки

ШИФР ГАММИРОВАНИЯ

10- Одноразовый блокнот Шеннона

ПОТОЧНЫЙ ШИФР

11 - A5/1

КОМБИНАЦИОННЫЙ ШИФР

12 - Кузнечик

АССИМЕТРИЧНЫЕ ШИФРЫ(ГЕНЕРАЦИЯ ЦИФРОВОЙ ПОДПИСИ)

13 - RSA

14 - Elgamal

15 - Обмен ключами по алгоритму Diffie-Hellman

3

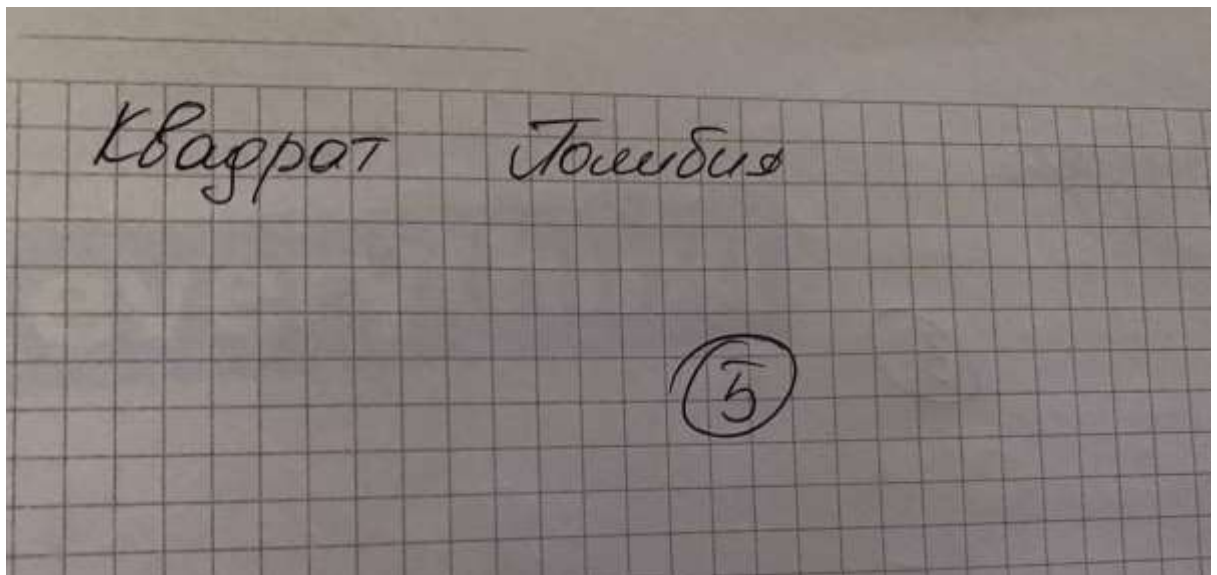
Зашифровано(Квадрат Полибия)

3116343511361533163234221642242332163324425641133424453563421633425126

Расшифровано(Квадрат Полибия)

леопарднеможетизменитьсясвоихпятентчк

Карточки:







## Блок В: ШИФРЫ МНОГОЗНАЧНОЙ ЗАМЕНЫ

### 4. Шифр Тритемия

Алгоритм шифра, описание:

$$Y_j = X_{i+j-1} \bmod n$$

**X** – исходный (открытый) текст

**Y** – зашифрованный текст

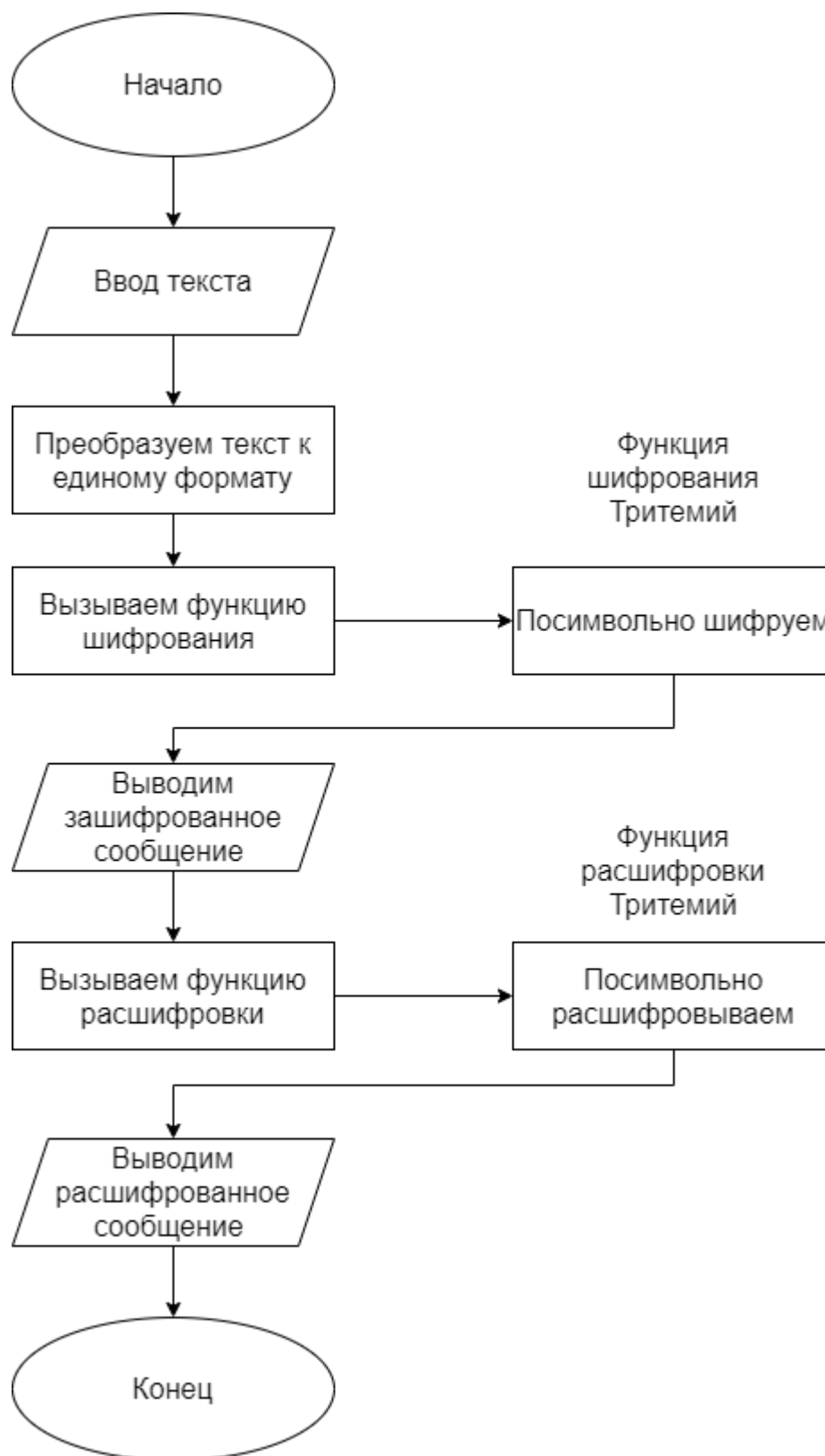
**i** – порядковый номер буквы в алфавите таблицы,  $i=1 \dots n$

**j** – порядковый номер буквы в тексте,  $j=1 \dots k$

**k** – количество букв в тексте

**n** – количество букв в выбранном алфавите (мощность алфавита).

Блок-схема программы:



Код программы с комментариями:

```
# ШИФР ТРИТЕЙЙ

def cryp_try(pr): #функция шифрования
    cryp_text = ""
```

```

for position, symb in enumerate(pr):
    k = position
    index = (alph.index(symb) + k) % len(alph) #ищем нужный символ
    cryp_text += alph[index]
return cryp_text

def dec_try(pr): #функция расшифровки
    decryp_text = ""
    for position, symb in enumerate(pr):
        k = position
        index = (alph.index(symb) - k) % len(alph) #ищем нужный символ
        decryp_text += alph[index]
    return decryp_text

# КОНЕЦ ШИФР ТРИТЕМИЙ

```

Часть кода с вызовом функций:

```

# Шифр Тритемия
elif code_cp == "5": # проверяем, какой шифр вызвал пользователь
    proverb = change_line(proverb) # вызываем функцию, которая преобразует строку

    cryp_text = cryp_try(proverb) # вызываем функцию шифрования
    print("Зашифровано(Шифр Тритемия)")
    print(cryp_text)

    decryp_text = dec_try(cryp_text) # вызываем функцию расшифровки
    print("Расшифровано(Шифр Тритемия)")
    print(decryp_text)

```

Тестирование:

Выберете шифр, которым хотели бы зашифровать:

ШИФРЫ ОДНОЗНАЧНОЙ ЗАМЕНЫ

1 - АТБАШ

2 - Шифр Цезаря

3 - Квадрат Полибия

ШИФРЫ МНОГОЗНАЧНОЙ ЗАМЕНЫ

4 - Шифр Белазо

5 - Шифр Тритемия

6 - Шифр Виженера

ШИФРЫ БЛОЧНОЙ ЗАМЕНЫ

7 - Шифр Плейфера

8 - Матричный шифр

ШИФР ПЕРСТАНОВКИ

9 - Шифр Вертикальной Перестановки

ШИФР ГАММИРОВАНИЯ

10- Одноразовый блокнот Шеннона

ПОТОЧНЫЙ ШИФР

11 - A5/1

КОМБИНАЦИОННЫЙ ШИФР

12 - Кузнечик

АССИМЕТРИЧНЫЕ ШИФРЫ(ГЕНЕРАЦИЯ ЦИФРОВОЙ ПОДПИСИ)

13 - RSA

14 - Elgamal

15 - Обмен ключами по алгоритму Diffie-Hellman

5

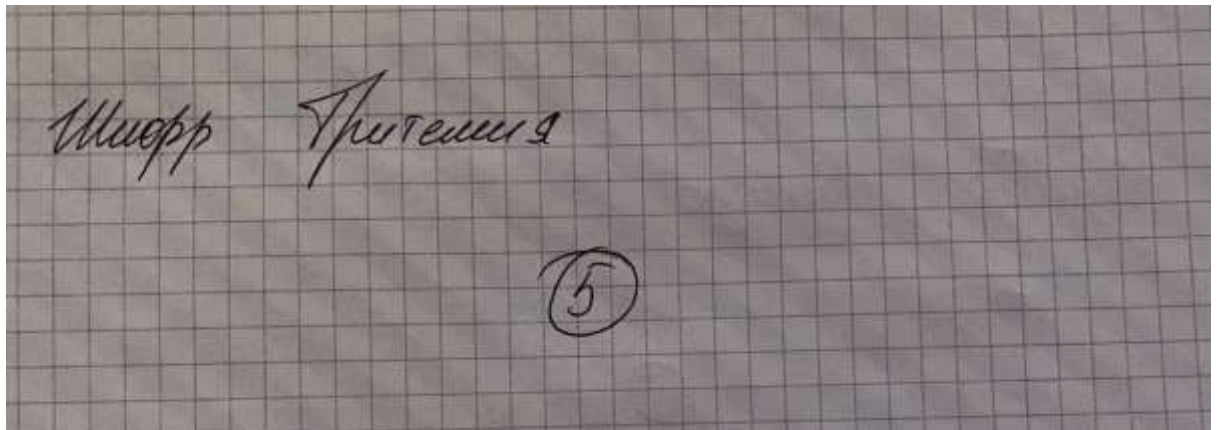
Зашифровано(Шифр Тритемия)

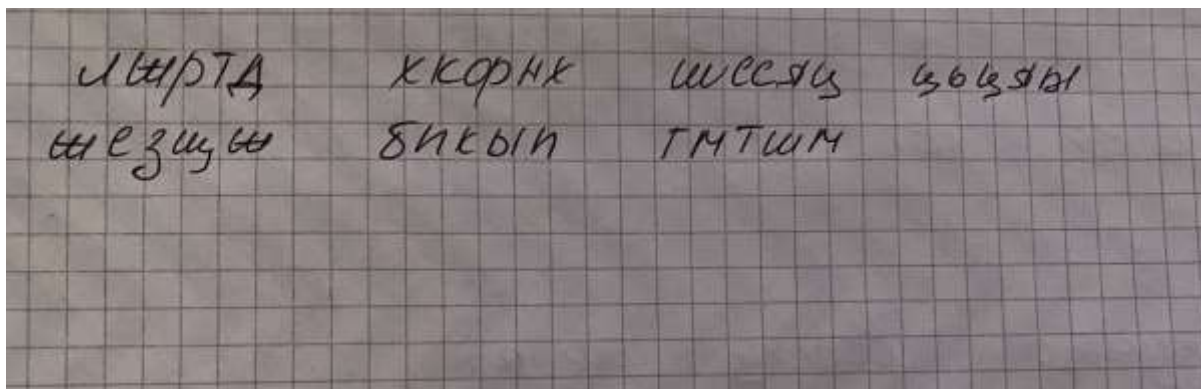
лжртдхкфнхшссяцъцъяжсзщбпкыпгмтшм

Расшифровано(Шифр Тритемия)

леопарднеможетиизменитьсясвоихпятентчк

Карточки:





Работа с текстом не менее 1000 знаков:

```

Попытка прерыва:
Попытка Терриллиана: Будничкин, Жит, да бей Будничкин, у него было ужо, шпале и перде. И он решил жениться. Он решил жениться, когда стукнет два пятачка да двести. Пошел в восемь он сделал предложение графе на 7 этаж. Графиня с собой согласилась немедленно, но в пятачки неукот двестиго его зносе и выдла шпале за водородный край. Дало было сделано, а графиня вернулась на стол к будничкину уже другим давай.
Был давайше неукот двестиго. Будничкин остался нит. Будничкин тогда сделал предложение очам. Очам было старе и неукотротно выдлае давай за ужо. Очам подумли пиль неукот и согласилась, но в этой момент им отчая давайе давай 30 ужо. Был уже восемь часов двадцать пять минут. Тогда будничкин быстро сделал предложение кимо: Кимо тут же согласилась, и будничкин стал давай, когда же стукнет бес пятачки да дв
ит. Сейчас его отец трампе католосся. Тут его отчая и накарал подувкой, потому что давайе зносе отчая. И бес пятачки давайе будничкин неукотрда для себя женикс на подувке.
выброси шпале, которая котеле бы закуривать.
(имя) (СКОРОЙ ЗАЯВКИ)
1 - АТАШ
2 - Аша Царя
3 - Евдарт Полюбин
(имя) (ПОСКОРОЙ ЗАЯВКИ)
4 - Аша Белька
5 - Аша Третьяков
6 - Аша Белька
(имя) (СКОРОЙ ЗАЯВКИ)
7 - Аша Полюбин
8 - Петрович аша
(имя) (ПРИСТАВКИ)
9 - Аша Вертикальной Перестановки
(имя) (ПРИСТАВКИ)
10 - Орловский Юлиант Зенкова
ПОСКОРОЙ ЗАЯВ
11 - АТА
12 - Аша Третьяков
13 - Аша
14 - Аша
15 - Аша
16 - Аша
17 - Аша
18 - Аша
19 - Аша
20 - Аша
21 - Аша
22 - Аша
23 - Аша
24 - Аша
25 - Аша
26 - Аша
27 - Аша
28 - Аша
29 - Аша
30 - Аша
31 - Аша
32 - Аша
33 - Аша
34 - Аша
35 - Аша
36 - Аша
37 - Аша
38 - Аша
39 - Аша
40 - Аша
41 - Аша
42 - Аша
43 - Аша
44 - Аша
45 - Аша
46 - Аша
47 - Аша
48 - Аша
49 - Аша
50 - Аша
51 - Аша
52 - Аша
53 - Аша
54 - Аша
55 - Аша
56 - Аша
57 - Аша
58 - Аша
59 - Аша
60 - Аша
61 - Аша
62 - Аша
63 - Аша
64 - Аша
65 - Аша
66 - Аша
67 - Аша
68 - Аша
69 - Аша
70 - Аша
71 - Аша
72 - Аша
73 - Аша
74 - Аша
75 - Аша
76 - Аша
77 - Аша
78 - Аша
79 - Аша
80 - Аша
81 - Аша
82 - Аша
83 - Аша
84 - Аша
85 - Аша
86 - Аша
87 - Аша
88 - Аша
89 - Аша
90 - Аша
91 - Аша
92 - Аша
93 - Аша
94 - Аша
95 - Аша
96 - Аша
97 - Аша
98 - Аша
99 - Аша
100 - Аша
101 - Аша
102 - Аша
103 - Аша
104 - Аша
105 - Аша
106 - Аша
107 - Аша
108 - Аша
109 - Аша
110 - Аша
111 - Аша
112 - Аша
113 - Аша
114 - Аша
115 - Аша
116 - Аша
117 - Аша
118 - Аша
119 - Аша
120 - Аша
121 - Аша
122 - Аша
123 - Аша
124 - Аша
125 - Аша
126 - Аша
127 - Аша
128 - Аша
129 - Аша
130 - Аша
131 - Аша
132 - Аша
133 - Аша
134 - Аша
135 - Аша
136 - Аша
137 - Аша
138 - Аша
139 - Аша
140 - Аша
141 - Аша
142 - Аша
143 - Аша
144 - Аша
145 - Аша
146 - Аша
147 - Аша
148 - Аша
149 - Аша
150 - Аша
151 - Аша
152 - Аша
153 - Аша
154 - Аша
155 - Аша
156 - Аша
157 - Аша
158 - Аша
159 - Аша
160 - Аша
161 - Аша
162 - Аша
163 - Аша
164 - Аша
165 - Аша
166 - Аша
167 - Аша
168 - Аша
169 - Аша
170 - Аша
171 - Аша
172 - Аша
173 - Аша
174 - Аша
175 - Аша
176 - Аша
177 - Аша
178 - Аша
179 - Аша
180 - Аша
181 - Аша
182 - Аша
183 - Аша
184 - Аша
185 - Аша
186 - Аша
187 - Аша
188 - Аша
189 - Аша
190 - Аша
191 - Аша
192 - Аша
193 - Аша
194 - Аша
195 - Аша
196 - Аша
197 - Аша
198 - Аша
199 - Аша
200 - Аша
201 - Аша
202 - Аша
203 - Аша
204 - Аша
205 - Аша
206 - Аша
207 - Аша
208 - Аша
209 - Аша
210 - Аша
211 - Аша
212 - Аша
213 - Аша
214 - Аша
215 - Аша
216 - Аша
217 - Аша
218 - Аша
219 - Аша
220 - Аша
221 - Аша
222 - Аша
223 - Аша
224 - Аша
225 - Аша
226 - Аша
227 - Аша
228 - Аша
229 - Аша
230 - Аша
231 - Аша
232 - Аша
233 - Аша
234 - Аша
235 - Аша
236 - Аша
237 - Аша
238 - Аша
239 - Аша
240 - Аша
241 - Аша
242 - Аша
243 - Аша
244 - Аша
245 - Аша
246 - Аша
247 - Аша
248 - Аша
249 - Аша
250 - Аша
251 - Аша
252 - Аша
253 - Аша
254 - Аша
255 - Аша
256 - Аша
257 - Аша
258 - Аша
259 - Аша
260 - Аша
261 - Аша
262 - Аша
263 - Аша
264 - Аша
265 - Аша
266 - Аша
267 - Аша
268 - Аша
269 - Аша
270 - Аша
271 - Аша
272 - Аша
273 - Аша
274 - Аша
275 - Аша
276 - Аша
277 - Аша
278 - Аша
279 - Аша
280 - Аша
281 - Аша
282 - Аша
283 - Аша
284 - Аша
285 - Аша
286 - Аша
287 - Аша
288 - Аша
289 - Аша
290 - Аша
291 - Аша
292 - Аша
293 - Аша
294 - Аша
295 - Аша
296 - Аша
297 - Аша
298 - Аша
299 - Аша
300 - Аша
301 - Аша
302 - Аша
303 - Аша
304 - Аша
305 - Аша
306 - Аша
307 - Аша
308 - Аша
309 - Аша
310 - Аша
311 - Аша
312 - Аша
313 - Аша
314 - Аша
315 - Аша
316 - Аша
317 - Аша
318 - Аша
319 - Аша
320 - Аша
321 - Аша
322 - Аша
323 - Аша
324 - Аша
325 - Аша
326 - Аша
327 - Аша
328 - Аша
329 - Аша
330 - Аша
331 - Аша
332 - Аша
333 - Аша
334 - Аша
335 - Аша
336 - Аша
337 - Аша
338 - Аша
339 - Аша
340 - Аша
341 - Аша
342 - Аша
343 - Аша
344 - Аша
345 - Аша
346 - Аша
347 - Аша
348 - Аша
349 - Аша
350 - Аша
351 - Аша
352 - Аша
353 - Аша
354 - Аша
355 - Аша
356 - Аша
357 - Аша
358 - Аша
359 - Аша
360 - Аша
361 - Аша
362 - Аша
363 - Аша
364 - Аша
365 - Аша
366 - Аша
367 - Аша
368 - Аша
369 - Аша
370 - Аша
371 - Аша
372 - Аша
373 - Аша
374 - Аша
375 - Аша
376 - Аша
377 - Аша
378 - Аша
379 - Аша
380 - Аша
381 - Аша
382 - Аша
383 - Аша
384 - Аша
385 - Аша
386 - Аша
387 - Аша
388 - Аша
389 - Аша
390 - Аша
391 - Аша
392 - Аша
393 - Аша
394 - Аша
395 - Аша
396 - Аша
397 - Аша
398 - Аша
399 - Аша
400 - Аша
401 - Аша
402 - Аша
403 - Аша
404 - Аша
405 - Аша
406 - Аша
407 - Аша
408 - Аша
409 - Аша
410 - Аша
411 - Аша
412 - Аша
413 - Аша
414 - Аша
415 - Аша
416 - Аша
417 - Аша
418 - Аша
419 - Аша
420 - Аша
421 - Аша
422 - Аша
423 - Аша
424 - Аша
425 - Аша
426 - Аша
427 - Аша
428 - Аша
429 - Аша
430 - Аша
431 - Аша
432 - Аша
433 - Аша
434 - Аша
435 - Аша
436 - Аша
437 - Аша
438 - Аша
439 - Аша
440 - Аша
441 - Аша
442 - Аша
443 - Аша
444 - Аша
445 - Аша
446 - Аша
447 - Аша
448 - Аша
449 - Аша
450 - Аша
451 - Аша
452 - Аша
453 - Аша
454 - Аша
455 - Аша
456 - Аша
457 - Аша
458 - Аша
459 - Аша
460 - Аша
461 - Аша
462 - Аша
463 - Аша
464 - Аша
465 - Аша
466 - Аша
467 - Аша
468 - Аша
469 - Аша
470 - Аша
471 - Аша
472 - Аша
473 - Аша
474 - Аша
475 - Аша
476 - Аша
477 - Аша
478 - Аша
479 - Аша
480 - Аша
481 - Аша
482 - Аша
483 - Аша
484 - Аша
485 - Аша
486 - Аша
487 - Аша
488 - Аша
489 - Аша
490 - Аша
491 - Аша
492 - Аша
493 - Аша
494 - Аша
495 - Аша
496 - Аша
497 - Аша
498 - Аша
499 - Аша
500 - Аша
501 - Аша
502 - Аша
503 - Аша
504 - Аша
505 - Аша
506 - Аша
507 - Аша
508 - Аша
509 - Аша
510 - Аша
511 - Аша
512 - Аша
513 - Аша
514 - Аша
515 - Аша
516 - Аша
517 - Аша
518 - Аша
519 - Аша
520 - Аша
521 - Аша
522 - Аша
523 - Аша
524 - Аша
525 - Аша
526 - Аша
527 - Аша
528 - Аша
529 - Аша
530 - Аша
531 - Аша
532
```

[illegible]

## Блок В: ШИФРЫ МНОГОЗНАЧНОЙ ЗАМЕНЫ

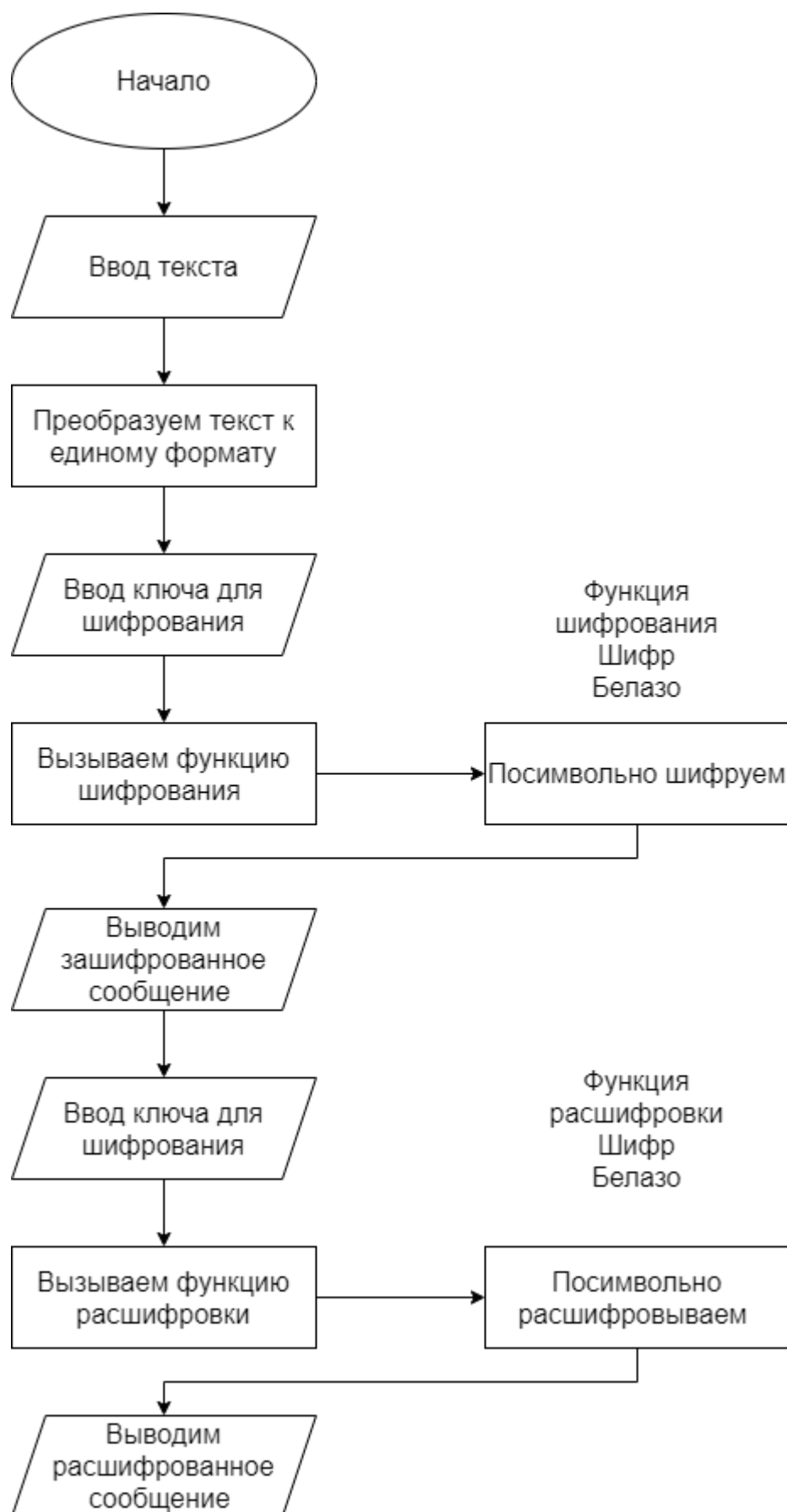
### 5. Шифр Белазо

Алгоритм шифра, описание:

Шифрование осуществляется с помощью пароля-ключа, состоящего из  $M$  символов. Из полной таблицы Тритемия выделяется матрица  $T_{\text{ш}}$  размерностью  $[(M+1) \times R]$ . Она включает первую строку и строки, первые элементы которых совпадают с символами ключа. Если в качестве ключа выбрано слово <ЗОНД>, то матрица шифрования содержит пять строк :

$T_{\text{в}} =$	А	Б	В	Г	Д	Е	Ж	З	И	К	.	.	.	Э	Ю	Я	␣
	З	И	К	Л	М	Н	О	П	Р	С	.	.	.	Г	Д	Е	Ж
	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	.	.	.	К	Л	М	Н
	Н	О	П	Р	С	Т	У	Ф	Х	Ц	.	.	.	И	К	Л	М
	Д	Е	Ж	З	И	К	Л	М	Н	О	.	.	.	А	Б	В	Г

Блок-схема программы:





## Код программы с комментариями:

```
# ШИФР БЕЛАЗО
def crypt_bel(pr, key): #функция шифрования
    crypt_text = ""
    for position, symb in enumerate(pr):
        k = alph.index(key[position % len(key)]) # ищем с какой позиции начинается
        index = (alph.index(symb) + k) % len(alph) # ищем нужный символ
        crypt_text += alph[index]
    return crypt_text

def dec_bel(pr, key): #функция расшифровки
    decryp_text = ""
    for position, symb in enumerate(pr):
        k = alph.index(key[position % len(key)]) # ищем с какой позиции начинается
        index = (alph.index(symb) - k) % len(alph) # ищем нужный символ
        decryp_text += alph[index]
    print(decryp_text)
    return decryp_text
# КОНЕЦ ШИФРА БЕЛАЗО
```

## Часть кода с вызовом функций:

```
# Белазо
elif code_cp == "4": # проверяем, какой шифр вызвал пользователь
    proverb = change_line(proverb) # вызываем функцию, которая преобразует строку
    print("Введите ключ для шифрования")
    key = input() # вводим ключ для шифрования
    crypt_text = crypt_bel(proverb, key) # вызываем функцию шифрования
    print("Зашифровано(Белазо)")
    print(crypt_text)
    print("Введите ключ для расшифровки")
    key = input() # вводим ключ для расшифровки
    decryp_text = dec_bel(crypt_text, key) # вызываем функцию расшифровки
    print("Расшифровано(Белазо)")
    print(decryp_text)
```

## Тестирование:

Выберете шифр, которым хотели бы зашифровать:

ШИФРЫ ОДНОЗНАЧНОЙ ЗАМЕНЫ

- 1 - АТБАШ
- 2 - Шифр Цезаря
- 3 - Квадрат Полибия

ШИФРЫ МНОГОЗНАЧНОЙ ЗАМЕНЫ

- 4 - Шифр Белазо
- 5 - Шифр Тритемия
- 6 - Шифр Виженера

ШИФРЫ БЛОЧНОЙ ЗАМЕНЫ

- 7 - Шифр Плейфера
- 8 - Матричный шифр

ШИФР ПЕРСТАНОВКИ

- 9 - Шифр Вертикальной Перестановки

ШИФР ГАММИРОВАНИЯ

- 10 - Одноразовый блокнот Шеннона

ПОТОЧНЫЙ ШИФР

- 11 - A5/1

КОМБИНАЦИОННЫЙ ШИФР

- 12 - Кузнечик

АССИМЕТРИЧНЫЕ ШИФРЫ(ГЕНЕРАЦИЯ ЦИФРОВОЙ ПОДПИСИ)

- 13 - RSA
- 14 - Elgamal
- 15 - Обмен ключами по алгоритму Diffie-Hellman

4

Введите ключ для шифрования

кот

Зашифровано(Белазо)

хуащовоычцъшпаъсччцджяфщцщндпыдбш

Введите ключ для расшифровки

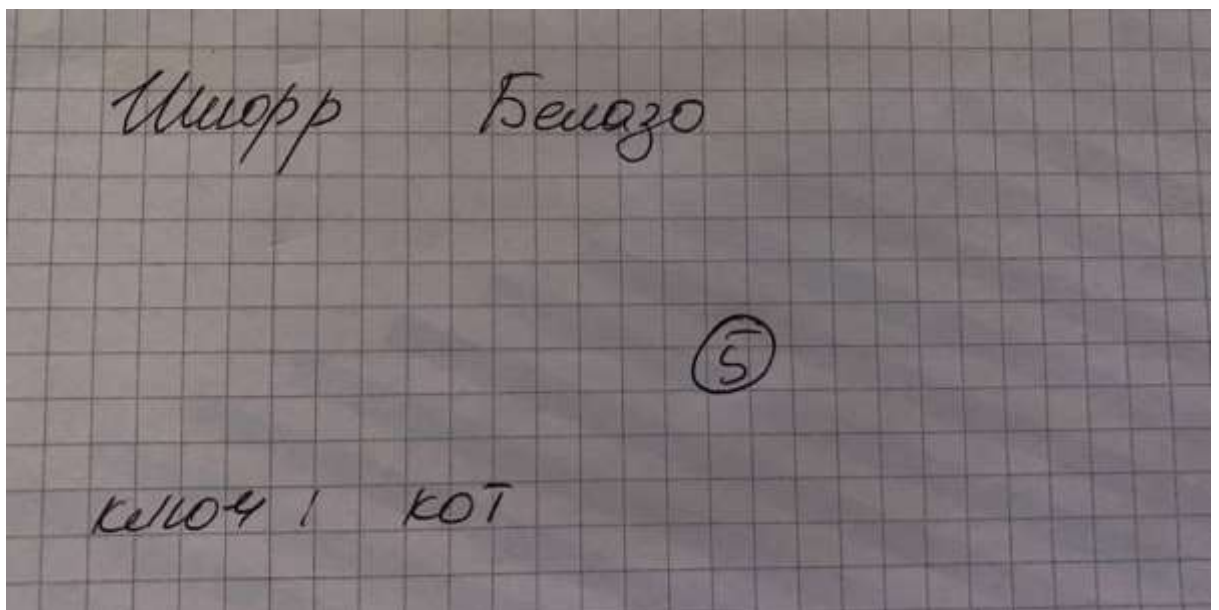
кот

леопарднеможетизменитьсясвоихпятентчк

Расшифрованно(Белазо)

леопарднеможетизменитьсясвоихпятентчк

Карточки:





## Блок В: ШИФРЫ МНОГОЗНАЧНОЙ ЗАМЕНЫ

### 6. Шифр Виженера

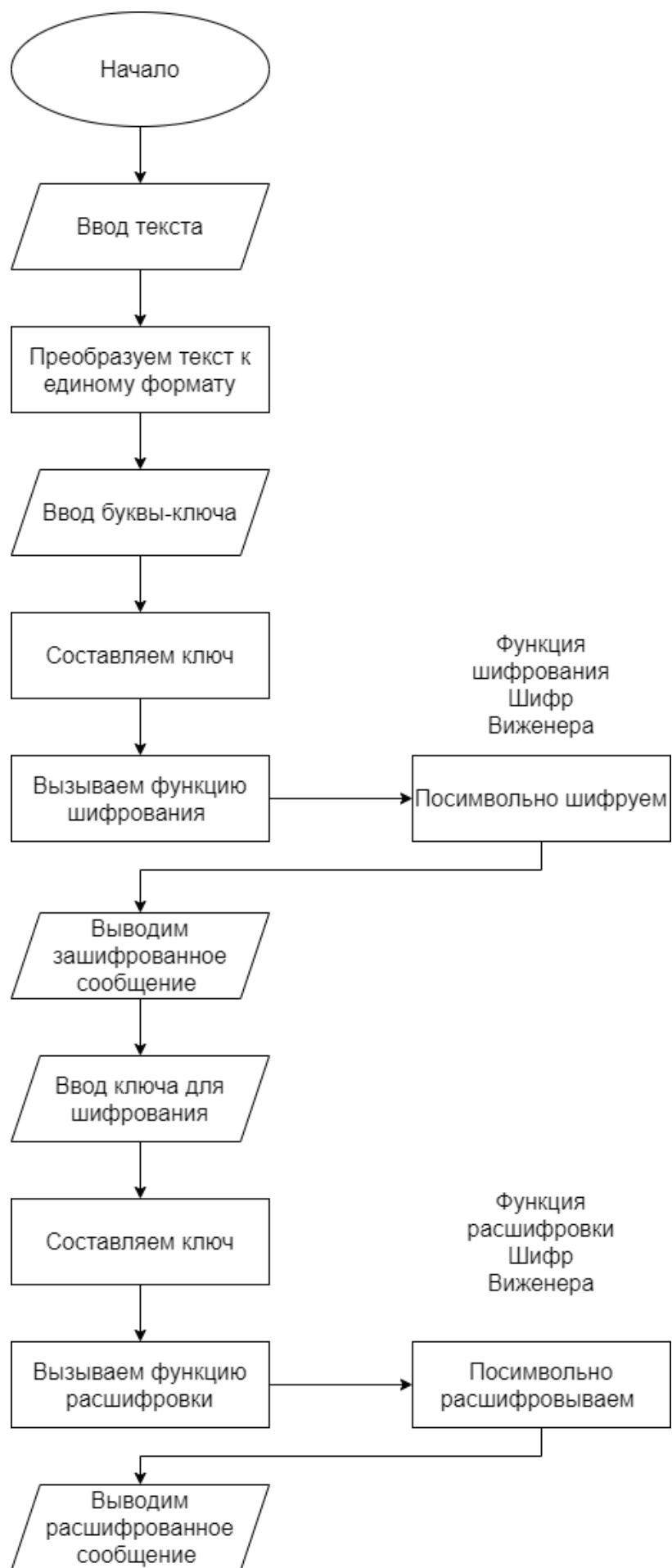
Алгоритм шифра, описание:

#### *2. Шифр ключом-шифртекстом*

$$\begin{array}{r} \Gamma = s_0s_1s_2...s_{i-1}... \\ + \quad T_O = t_1t_2t_3...t_i... \\ \hline T_{ш} = s_1s_2s_3...s_i... \end{array}$$

$s_0$  - секретная буква-ключ

Блок-схема программы:



## Код программы с комментариями:

```
# ШИФР ВИЖЕНЕРА

def cryp_vig(pr, key): #функция шифрования
    cryp_text = ""
    for position, symb in enumerate(pr):
        k = alph.index(key[position])
        index = (alph.index(symb) + k) % len(alph) #ищем нужный символ
        cryp_text+= alph[index]
    return cryp_text

def dec_vig(pr, key): #функция расшифровки
    decryp_text = ""
    for position, symb in enumerate(pr):
        k = alph.index(key[position])
        index = (alph.index(symb) - k) % len(alph) #ищем нужный символ
        decryp_text += alph[index]
    return decryp_text

# КОНЕЦ ШИФР ВИЖЕНЕРА
```

## Часть кода с вызовом функций:

```
elif code_cp == "6": # проверяем, какой шифр вызвал пользователь
    proverb = change_line(proverb) # вызываем функцию, которая преобразует строку
    n = len(proverb)
    print("Введите ключ для шифрования")
    key = input() # вводим ключ для шифрования
    key += proverb[n-1]
    cryp_text = cryp_vig(proverb, key) # вызываем функцию шифрования
    print("Зашифровано(Шифр Виженера)")
    print(cryp_text)
    n = len(proverb)
    print("Введите ключ для шифрования")
    key = input() # вводим ключ для расшифровки
    key += proverb[n-1]
    decryp_text = dec_vig(cryp_text, key) # вызываем функцию расшифровки
    print("Расшифровано(Шифр Виженера)")
    print(decryp_text)
```

## Тестирование:

Выберете шифр, которым хотели бы зашифровать:

ШИФРЫ ОДНОЗНАЧНОЙ ЗАМЕНЫ

1 - АТБАШ

2 - Шифр Цезаря

3 - Квадрат Полибия

ШИФРЫ МНОГОЗНАЧНОЙ ЗАМЕНЫ

4 - Шифр Белазо

5 - Шифр Тритемия

6 - Шифр Виженера

ШИФРЫ БЛОЧНОЙ ЗАМЕНЫ

7 - Шифр Плейфера

8 - Матричный шифр

ШИФР ПЕРСТАНОВКИ

9 - Шифр Вертикальной Перестановки

ШИФР ГАММИРОВАНИЯ

10- Одноразовый блокнот Шеннона

ПОТОЧНЫЙ ШИФР

11 - A5/1

КОМБИНАЦИОННЫЙ ШИФР

12 - Кузнечик

АССИМЕТРИЧНЫЕ ШИФРЫ(ГЕНЕРАЦИЯ ЦИФРОВОЙ ПОДПИСИ)

13 - RSA

14 - Elgamal

15 - Обмен ключами по алгоритму Diffie-Hellman

6

Введите ключ для шифрования

к

Зашифровано(Шифр Виженера)

хруэпрфстсъфлчьпустхьонурцэдосчтяйб

Введите ключ для шифрования

к

Расшифровано(Шифр Виженера)

леопарднеможетиизменитьсясвоихпятентчк

PS: C:\Users\Boriska\Downloads\Белазо\Белазо\Ключи.txt ■

Карточки:

Иванов Виталий

⑤

ключ: К

хруст рррррррррррррррррррррр  
у издого чдддд





## Блок С: ШИФРЫ БЛОЧНОЙ ЗАМЕНЫ

### 8. Матричный шифр

Алгоритм шифра, описание:

Предполагается, что буквы занумерованы от 0 до 32 и рассматриваются как элементы некоторого алгебраического кольца. Если к  $n$ -грамме сообщения

применить матрицу  $a_{ij}$ , то получится  $n$ -грамма криптограммы

$$l_i = \sum_{j=1}^n a_{ij} m_j, \quad i = 1, \dots, n.$$

Матрица  $a_{ij}$  является ключом, и расшифровка выполняется с помощью обратной матрицы.

Обратная матрица будет существовать тогда и только тогда, когда определитель  $|a_{ij}|$  имеет обратный элемент в кольце.

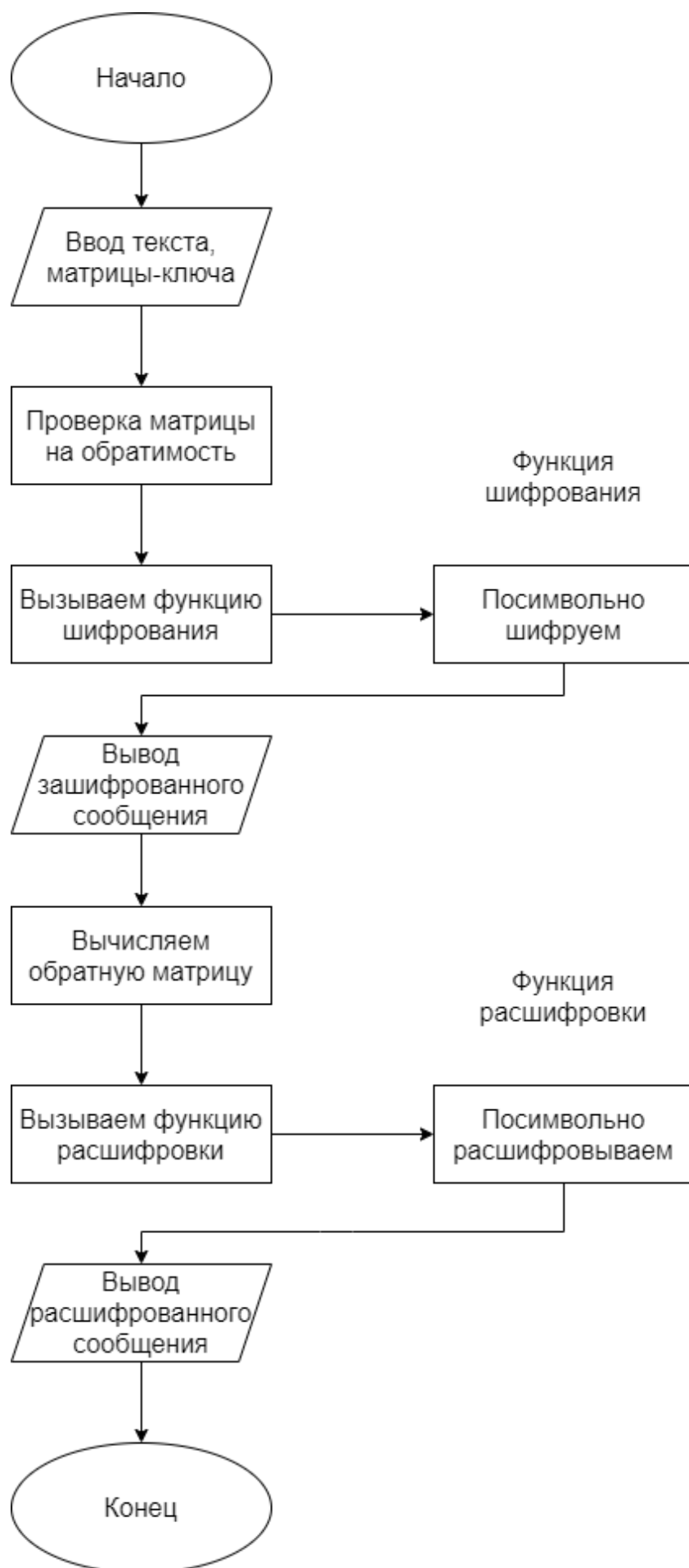
В качестве матричного шифрования информации могут использоваться аналитические преобразования, основанные, на преобразованиях матричной алгебры.

Шифрование  $k$ -ого блока исходной информации, представленного в виде вектора  $B_k = \|b_j\|$ , осуществляется путем перемножения этого вектора на матрицу  $A = \|a_{ij}\|$ , используемую в качестве ключа. В результате перемножения получается блок шифротекста в виде вектора  $C_k = \|c_i\|$ , где элементы вектора  $C_k$  определяются по формуле:

$$C_i = \sum_j a_{ij} b_j$$

Блок-схема

программы:



## Код программы с комментариями:

```
import numpy as np
import math
from numpy import linalg as inverse

alphabet = "0абвгдеёжзийклмнопрстуфхцчщъыьэюя"

def inverse_matr(matrix): # Проверка матрицы на обратимость
    try:
        in_matr = inverse.inv(matrix) # вычисляем обратную матрицу
    except p.linalg.LinAlgError:
        in_matr = "Ошибка, матрица необратима" # обрабатываем ошибку, если обратной матрицы не
        существует
    return in_matr

def cryp_matr(pr, matrkey): #функция шифрования
    n = len(pr)
    k = math.ceil(n / 5)
    i = 0

    s = n % 5
    if s != 0:
        pr = pr + (5 - s) * "0"

    text = pr
    cryp_text = ""

    for i in range(k):
        matr = p.matrix([[alphabet.find(text[0])], [alphabet.find(text[1])],
        [alphabet.find(text[2])], [alphabet.find(text[3])], [alphabet.find(text[4])]])
        text = text[5:]
        matrcryp = p.dot(matrkey, matr)
        # print (matrcryp)

        s1 = str(matrcryp[0][0])
        s1 = s1 [s1.find("[") + 1:s1.find("]") + 1]
        s1 = s1 [s1.find("[") + 1:s1.find(")")]

        s2 = str(matrcryp[1][0])
        s2 = s2 [s2.find("[") + 1:s2.find("]") + 1]
        s2 = s2 [s2.find("[") + 1:s2.find(")")]

        s3 = str(matrcryp[2][0])
        s3 = s3 [s3.find("[") + 1:s3.find("]") + 1]
        s3 = s3 [s3.find("[") + 1:s3.find(")")]

        s4 = str(matrcryp[3][0])
        s4 = s4 [s4.find("[") + 1:s4.find("]") + 1]
        s4 = s4 [s4.find("[") + 1:s4.find(")")]

        s5 = str(matrcryp[4][0])
        s5 = s5 [s5.find("[") + 1:s5.find("]") + 1]
        s5 = s5 [s5.find("[") + 1:s5.find(")")]

        cryp_text = cryp_text + s1 + " " + s2 + " " + s3 + " " + s4 + " " + s5 + " "
```

```

return cryp_text

def dec_matr(pr, matrkey): #функция расшифровки
    decryp_text = ""
    pr = pr.split()
    i = 0
    n = math.ceil(len(pr) / 5)
    s = len(pr) % 5
    pr_1 = []
    if s != 0:
        for i in range(s):
            pr.append(0)
    i = 0

    for i in range(n):
        matr = p.matrix([[float(pr[0])], [float(pr[1])], [float(pr[2])], [float(pr[3])],
[float(pr[4])]])
        # matr = p.matrix([[pr[0]], [pr[1]], [pr[2]], [pr[3]], [pr[4]]])
        k = len(pr)
        j = 0
        pr_1 = []
        for j in range(k - 5):
            pr_1.append(pr[j+5])
        pr = pr_1
        matrdec = p.dot(matrkey, matr)
        a = 0
        for a in range(5):
            decryp_text += alphabet[int(matrdec[a][0])]
    return decryp_text

# proverb = "Леопард не может изменить своих пятен."
# proverb = " Людмила Петрушевская. Будильник. Жил, да был будильник. У него были усы,
шляпа и сердце. И он решил жениться. Он решил жениться, когда стукнет без пятнадцати
девять. Ровно в восемь он сделал предложение графину с водой. Графин с водой согласился
немедленно, но в пятнадцать минут девятого его унесли и выдали замуж за водопроводный кран.
Дело было сделано, и графин вернулся на стол к будильнику уже замужней дамой. Было двадцать
минут девятого. Времени оставалось мало. Будильник тогда сделал предложение очкам. Очки
были старые и неоднократно выходили замуж за уши. Очки подумали пять минут и согласились,
но в этот момент их опять выдали замуж за уши. Было уже восемь часов двадцать пять минут.
Тогда будильник быстро сделал предложение книге. Книга тут же согласилась, и будильник стал
ждать, когда же стукнет без пятнадцати девять. Сердце его очень громко колотилось. Тут его
взяли и накрыли подушкой, потому что детей уложили спать. И без пятнадцати девять будильник
неожиданно для себя женился на подушке."
proverb = input("Введите текст для шифрования: ")

proverb = proverb.replace(" ", "")
proverb = proverb.lower()

while proverb.find(",") != -1:
    str1 = proverb[:proverb.find(",")]
    str2 = proverb[proverb.find(",")+1:]
    proverb = str1 + "зпт" + str2

while proverb.find(".") != -1:
    str1 = proverb[:proverb.find(".")]
    str2 = proverb[proverb.find(".")+1:]
    proverb = str1 + "тчк" + str2

```

```

# matrkey_c = p.matrix([[9.86, 3.52, 12.61, 8.00, 4.48], [5.78, 2.18, -1, 1, 1], [3, 1, 1, 1, 1], [2, 1, 1, 4, 1], [2, -1, 1, 1, 5]])
matrkey_c = p.matrix([[4, 1, 1, 2, 1], [1, 2, -1, 1, 1], [3, 1, 1, 1, 1], [2, 1, 1, 4, 1], [2, -1, 1, 1, 5]]) # матрица шифрования
# matrkey_c = p.matrix([[ 1, 2], [2, 4]])
print("Матрица шифрования:")
print(matrkey_c)
matrkey_d = inverse_matr(matrkey_c) # проверяем матрицу на обратимость и сразу же вычисляем обратную матрицу
crypt_text = crypt_matr(proverb, matrkey_c) # шифруем текст
print(crypt_text)
print("Матрица расшифровки:")
print(matrkey_d)
decrypt_text = dec_matr(crypt_text, matrkey_d) # расшифровываем текст
print(decrypt_text)

```

## Тестирование:

```

Введите текст для шифрования: Леопард не может изменить своих пятен.
Матрица шифрования:
[[ 4 1 1 2 1]
 [ 1 2 -1 1 1]
 [ 3 1 1 1 1]
 [ 2 1 1 4 1]
 [ 2 -1 1 1 5]]
189 27 79 117 58 118 33 94 94 122 128 56 92 136 189 96 56 72 188 75 151 88 128 117 112 166 92 123 212 147 121 53 98 159 182
Матрица расшифровки:
[[ 0.75 0. -0.5 -0.25 0. ]
 [-0.96875 0.25 1.1875 0.15625 -0.125 ]
 [-1.234375 -0.375 1.59375 0.328125 -0.0625 ]
 [ 0.25 0. -0.5 0.25 0. ]
 [-0.296875 0.125 0.21875 0.015625 0.1875 ]]
леопард не может изменить своих пятен

```

## Карточки:





Работа с текстом не менее 1000 знаков:

Введите текст для шифрования: Людмила Петрушевская. Буди́льник. Жил, да был буди́льник. У него были усы, шляпа и сердце. И он решил жениться. Он решил жениться, когда стукнет без пятнадцати девять. Ровно в восемь он сделал предложение графине с водой. Графин с водой согласился немедленно, но в пятнадцать минут девятого его унесли и выдали замуж за водопроводный кран. Дело было сделано, и графин вернулся на стол к будильнику уже замужней дамой. Было двадцать минут девятого. Времени оставалось мало. Буди́льник тогда сделал предложение очкам. Очки были старые и неоднократно вывалили замуж за усы. Очки подумали пять минут и согласились, но в этот момент их опять выдали замуж за усы. Было уже восемь часов двадцать пять минут. Тогда буди́льник быстро сделал предложение книге. Книга тут же согласилась, и буди́льник стал ждать, когда же стукнет без пятнадцати девять. Сердце его очень громко колотилось. Тут его в затылок накрыли подушкой, потому что детей уложили спать. И без пятнадцати девять буди́льник неожиданно для себя женился на подушке.

Матрица шифрования:

```
[[ 4 1 1 3 1]
 [ 1 2 -1 1 1]
 [ 3 1 1 1 1]
 [ 2 1 1 4 1]
 [ 2 -1 1 1 5]]
```

127 96 100 129 63 102 24 83 88 148 134 43 110 110 62 175 95 123 203 166 161 73 115 153 86 123 31 98 133 102 125 47 101 125 81 116 64 89 130 133 94 47 60  
142 105 67 62 55 83 63 100 82 147 153 167 109 49 120 140 104 105 76 72 155 88 147 71 108 181 03 173 77 143 165 218 123 34 87 127 02 133 38 109 100 101  
169 70 129 151 130 137 27 109 121 131 93 35 75 97 135 237 80 187 217 219 121 53 91 133 71 149 47 115 113 93 169 64 124 109 165 210 66 157 104 154 94 30  
77 64 129 149 71 114 139 76 123 44 94 181 138 174 64 140 110 87 142 21 108 114 102 130 29 184 158 212 109 92 131 105 125 52 23 46 52 25 147 92 117 143 1  
83 114 38 93 92 71 118 37 88 126 145 82 27 60 182 76 85 36 53 65 33 130 23 102 114 142 143 77 106 133 67 157 55 110 175 120 89 71 63 125 63 119 71 88 12  
1 55 130 39 98 130 76 105 19 82 00 110 150 90 125 124 96 69 24 57 69 101 139 75 107 143 140 146 94 114 150 109 130 78 105 100 151 92 35 77 118 76 117 48  
87 97 64 101 69 82 127 29 127 61 95 127 151 121 31 93 117 100 83 -7 08 73 56 87 39 73 63 96 106 32 84 68 59 137 45 104 139 150 100 26 87 86 125 176 58  
120 154 82 114 47 107 128 72 86 47 78 78 102 183 37 85 87 64 100 25 80 115 161 124 60 103 98 118 60 53 53 87 20 112 81 85 142 95 145 90 125 109 116 135  
53 103 119 59 155 64 121 139 210 145 65 109 157 158 63 26 54 40 90 130 39 103 100 110 78 20 57 100 94 150 86 128 114 174 88 44 64 60 23 115 102 90 145 1  
57 143 60 100 137 154 115 67 77 171 140 149 53 111 157 189 95 20 77 83 115 100 61 78 120 118 99 33 79 61 100 142 39 112 138 62 150 62 112 174 115 67 62  
95 83 63 180 82 147 153 167 94 30 77 64 120 54 18 48 40 83 115 65 93 01 02 118 51 87 116 95 90 26 72 102 38 141 57 115 117 125 155 70 120 109 195 122 35  
80 136 60 142 95 114 126 88 91 64 70 180 96 122 82 90 138 102 150 32 119 172 161 72 31 57 82 60 64 25 55 78 62 93 47 80 109 117 190 78 140 100 130 104  
54 89 94 129 167 30 83 90 126 220 67 173 182 148 148 66 115 130 134 162 31 85 72 137 131 75 102 140 186 119 54 95 131 116 115 81 90 141 111 113 53 93 97  
107 162 50 126 154 154 217 75 181 157 224 63 13 48 73 77 64 25 53 78 62 91 47 80 109 117 181 80 131 193 134 108 59 86 88 48 147 92 117 143 183 155 27 1  
14 137 99 58 30 48 58 133 121 61 103 153 194 109 91 129 139 109 186 78 140 104 127 111 54 86 81 38 67 62 55 83 63 171 64 129 135 77 207 81 160 185 157 1  
14 37 82 182 57 104 40 85 90 58 103 58 84 80 90 98 35 73 100 91 104 33 75 146 187 82 37 66 86 28 143 56 115 119 77 123 61 91 111 44 120 46 100 84 137 19  
6 61 146 170 110 67 62 55 83 63 180 81 140 152 162 115 22 87 91 85 80 37 79 105 186 121 37 97 89 73 76 50 50 112 119 143 56 120 113 151 90 55 71 120 189  
130 78 105 100 151 50 42 44 58 27 145 99 112 199 123 161 54 130 123 153 66 57 55 66 18 132 44 110 112 122 188 50 142 160 160 121 61 93 129 101 136 75 1  
67 130 115 188 96 144 200 113 137 32 111 109 65 155 59 100 189 130 79 28 60 61 96 151 90 123 135 115 154 43 112 174 134 138 66 105 140 147 150 59 117 14  
8 124 106 72 129 136 78 03 72 78 103 114 109 60 80 103 93 98 52 87 80 119 109 67 137 163 127 90 55 71 120 189 130 78 105 100 51 50 42 44 58 27 127 81 9  
4 181 33 155 64 121 139 210 118 44 88 110 77 93 28 72 71 44 134 47 94 94 101 104 100 159 132 120 86 28 68 90 100 143 88 123 107 106 154 43 112 174 134 9  
3 33 75 185 29

Матрица расшифровки:

```
[[ 0.75 0. -0.5 -0.25 0. ]
 [-0.96875 0.25 1.1875 0.15625 -0.125 ]
 [-1.234375 -0.375 1.59375 0.328125 -0.0625 ]
 [ 0.25 0. -0.5 0.25 0. ]
 [-0.296875 0.125 0.21875 0.015625 0.1875 ]]
```

Введите текст для дешифрования: Людмила Петрушевская. Буди́льник. Жил, да был буди́льник. У него были усы, шляпа и сердце. И он решил жениться. Он решил жениться, когда стукнет без пятнадцати девять. Ровно в восемь он сделал предложение графине с водой. Графин с водой согласился немедленно, но в пятнадцать минут девятого его унесли и выдали замуж за водопроводный кран. Дело было сделано, и графин вернулся на стол к будильнику уже замужней дамой. Было двадцать пять минут девятого. Времени оставалось мало. Буди́льник тогда сделал предложение очкам. Очки были старые и неоднократно вывалили замуж за усы. Очки подумали пять минут и согласились, но в этот момент их опять выдали замуж за усы. Было уже восемь часов двадцать пять минут. Тогда буди́льник быстро сделал предложение книге. Книга тут же согласилась, и буди́льник стал ждать, когда же стукнет без пятнадцати девять. Сердце его очень громко колотилось. Тут его в затылок накрыли подушкой, потому что детей уложили спать. И без пятнадцати девять буди́льник неожиданно для себя женился на подушке.



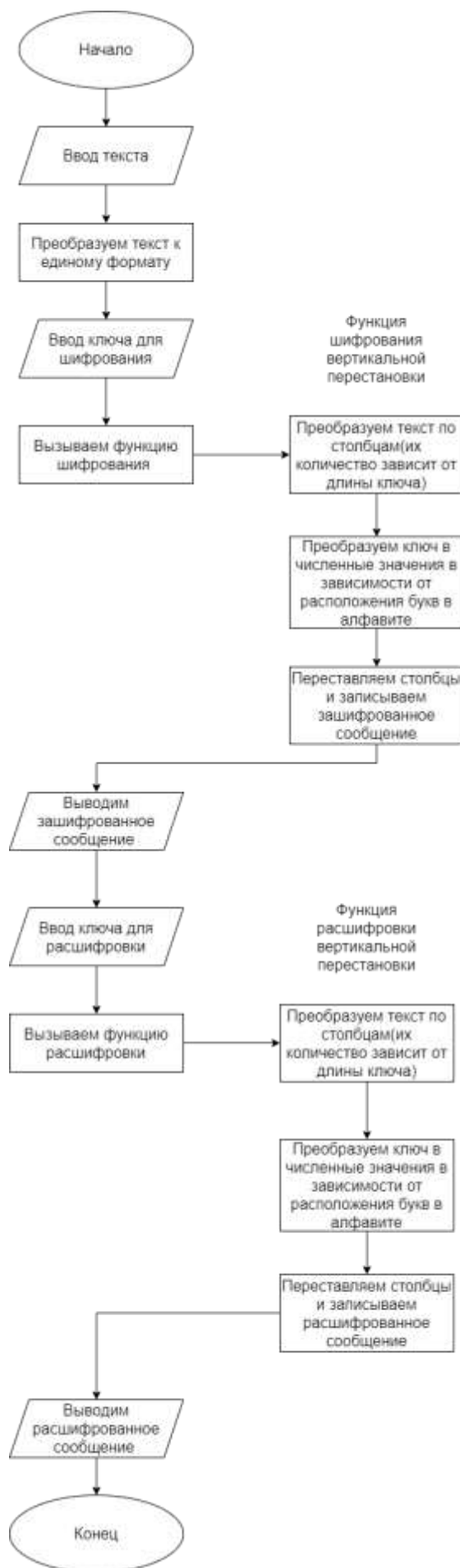
## **Блок D: ШИФРЫ ПЕРЕСТАНОВКИ**

### **10. Шифр Вертикальной Перестановки**

Алгоритм шифра, описание:

Можно записать исходное сообщение в прямоугольную матрицу, выбрав такой маршрут: по горизонтали, начиная с левого верхнего угла поочередно слева направо и справа налево. Считывать же шифрованное сообщение по другому маршруту: по вертикали, начиная с правого верхнего угла и двигаясь поочередно сверху вниз и снизу вверх.

Блок-схема программы:



## Код программы с комментариями:

```
# ВЕРТИКАЛЬНАЯ ПЕРЕСТАНОВКА
def сгрупп_вер(pr,k): #функция шифрования
    i = 0
    n_k = len(k) # смотрим длину ключевого слова
    if len(pr) % n_k != 0: # считаем количество строк
        n = len(pr)//n_k + 1
    else:
        n = len(pr)//n_k
    text = pr
    matr = []
    for i in range(n): # разделяем текст на строки
        if i % 2 != 0:
            str1 = text [:n_k]
            matr.append(str1[::-1]) #записываем строку в обратном порядке, так как при шифровании
            текст записывается "змейкой"
            text = text [n_k:]
        else:
            matr.append(text [:n_k])
            text = text [n_k:]

    i = 0
    n = len(matr[1])
    m = len(matr)
    if len(matr[m-1]) != n: # последнюю строку дозаполняем прочерками для удобства обработки
        matr[m-1] = matr[m-1] + "-" * (n - len(matr[m-1]))
    column = []

    for i in range(n): #заполняем столбцы
        word = ""
        for j in range(m):
            word = word + matr[j][i]
        column.append(word)

    k2 = list(k)
    j = 0
    for i in alph: #инденсируем буквы ключа
        if k.find(i) != -1:
            k2[k.find(i)] = j
            j += 1
    result = ''
    for i in k2: #переставляем столбцы
        result = result + column[int(i)]
    return result

def dec_ver(pr,k): #функция расшифровки
    k2 = list(k)
    j = 0
    for i in alph: # индексируем ключ
        if k.find(i) != -1:
            k2[k.find(i)] = j
            j += 1

    k_symb = len(pr) / len(k) #смотрим сколько целых столбцов у нас есть
    k_symb_b = len(pr) % len(k) #смотрим сколько неполных столбцов
    column=[]
    pr2 = pr
    while pr2 != "": # разделяем на столбцы
```

```

        column.append(pr2[0:k_symb])
        pr2=pr2[k_symb:]

column2 = []
n_k2 =len (k2)
for i in range(n_k2): #переставляем столбцы
    column2.append(column[int(k2[i])])
matr = []
n = len(column2[1])
m = len(column2)

for i in range(n): #собираем текст
    word = ""
    for j in range(m):
        if i % 2 == 0:
            word = word + str(column2[j][i])
        else:
            word = word + str(column2[len(k) - 1- j][i])
    matr.append(word)

result = ""
n_matr = len(matr)
for i in range(n_matr):
    result = result + str(matr[i])

return result
# КОНЕЦ ВЕРТИКАЛЬНАЯ ПЕРЕСТАНОВКА

```

Часть кода с вызовом функций:

```

elif code_cp == "9": # проверяем, какой шифр вызвал пользователь
    proverb = change_line(proverb) # вызываем функцию, которая преобразует строку
    print("Введите ключ для шифрования")
    key = input() # вводим ключ для шифрования
    crypt_text = crypt_ver(proverb, key) # вызываем функцию шифрования
    print("Зашифровано(Шифр Вертикальной Перестановки)")
    print(crypt)
    print("Введите ключ для расшифровки")
    key = input() # вводим ключ для расшифровки
    decrypt_text = dec_ver(crypt_text, key) # вызываем функцию расшифровки
    print("Расшифровано(Шифр Вертикальной Перестановки)")
    print(decrypt_text)

```

Тестирование:

8 - Матричный шифр  
 ШИФР ПЕРСТАНОВКИ  
 9 - Шифр Вертикальной Перестановки  
 ШИФР ГАММИРОВАНИЯ  
 10- Одноразовый блокнот Шеннона  
 ПОТОЧНЫЙ ШИФР  
 11 - A5/1  
 КОМБИНАЦИОННЫЙ ШИФР  
 12 - Кузнечик  
 АССИМЕТРИЧНЫЕ ШИФРЫ(ГЕНЕРАЦИЯ ЦИФРОВОЙ ПОДПИСИ)  
 13 - RSA  
 14 - Elgamal  
 15 - Обмен ключами по алгоритму Diffie-Hellman

9  
 Введите ключ для шифрования  
 бактерия  
 Зашифровано(Шифр Вертикальной Перестановки)  
 ееечлзмнтажтпдмсиотнткроехпеиянево  
 Введите ключ для расшифровки  
 бактерия  
 Расшифровано(Шифр Вертикальной Перестановки)  
 леопарднеможетизменитьсвоихпятентчк  
 D:\Users\Получа\Desktop\5 семестр\Криптоз ■

Карточки:

Шифр вертикальной перестановки

(5)

ключ : бактерия

ееееч лзмнт антпг мейот  
 иткро вхпей янебо

Работа с текстом не менее 1000 знаков:

Видите клен для шифрования  
бактерия

Расшифровано (Info Вертикальной Парестановки)

Видите клен для расшифровки  
бактерия

Расшифровано (Info Вертикальной Парестановки)

## Блок Е: ШИФРЫ ГАММИРОВАНИЯ

### 13. Одноразовый блокнот Шеннона

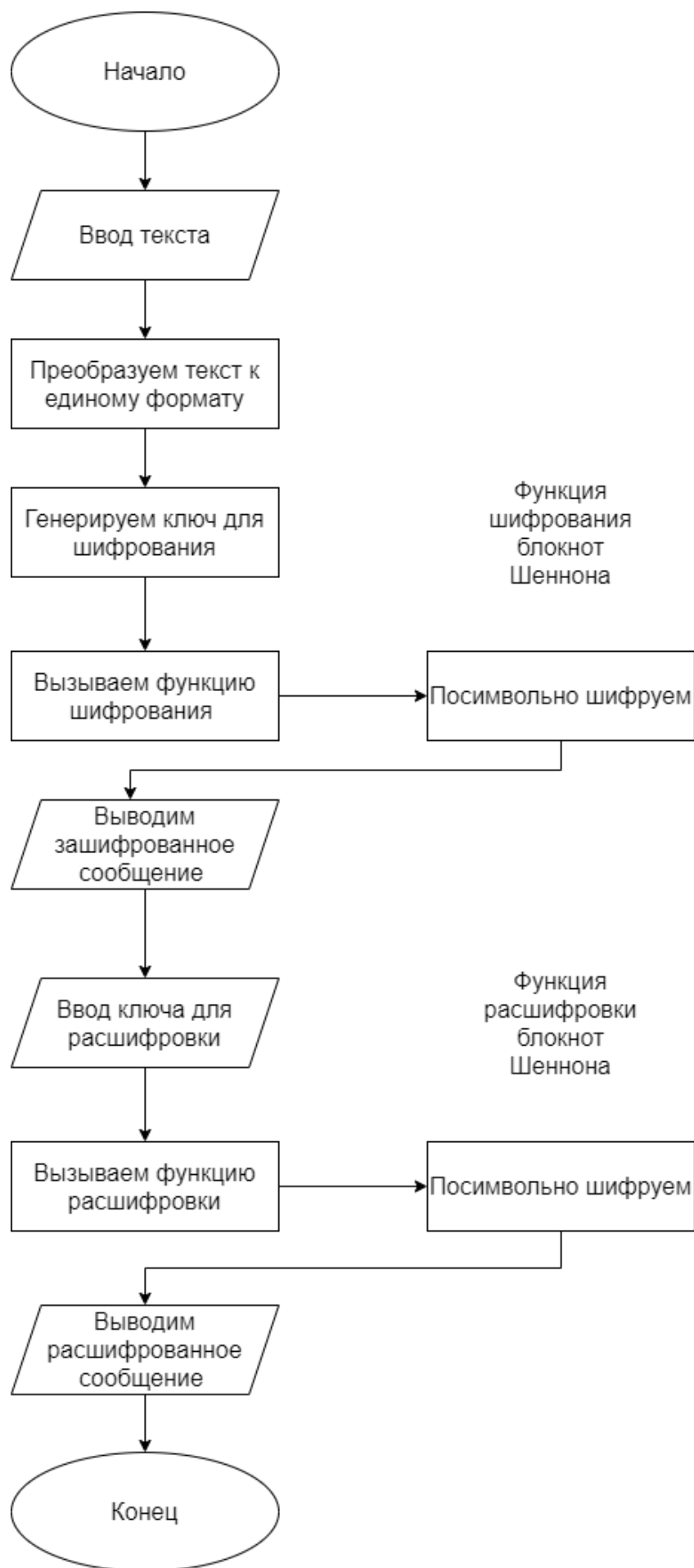
Алгоритм шифра, описание:

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

Шифртекст  $c = c_0c_1\dots c_{n-1}$  определяется соотношением  $c_i = m_i \oplus k_i$  при  $0 \leq i \leq n-1$ , где  $\oplus$  обозначает операцию «исключающее ИЛИ» (ассемблерная операция XOR) по модулю два или по любому другому модулю в случае символьной гаммы.

В своей исторической работе «Communication theory of secrecy systems» («Теория связи в секретных системах», 1949г.) Шеннон доказал то, что **одноразовый гамма-блокнот является «невскрываемой» шифрсистемой.**

Блок-схема программы:





## Код программы с комментариями:

```
# БЛОКНОТ ШЕННОНА
def cryp_sh(pr, key): #функция шифрования
    cryp_text = ""
    for m, k in zip(pr, range(len(key))):
        cryp_text = cryp_text + alph[(alph.find(m)+int(key[k]))%32] #ищем нужный символ
    return cryp_text

def dec_sh(pr,key): #функция расшифровки
    decryp_text = ""
    for m, k in zip(pr, range(len(key))):
        decryp_text = decryp_text + alph[(alph.find(m)-int(key[k]))%32] #ищем нужный символ
    return decryp_text
# КОНЕЦ БЛОКНОТ ШЕННОНА
```

## Часть кода с вызовом функций:

```
# Одноразовый блокнот Шеннона
elif code_cp == "10": # проверяем, какой шифр вызвал пользователь
    proverb = change_line(proverb) # вызываем функцию, которая преобразует строку

    print("Генерируем ключ для шифрования")
    key=[random.randint(1,32) for i in range(len(proverb))] #генеринуем ключ
    print(key)
    cryp_text = cryp_sh(proverb, key) # вызываем функцию шифрования
    print("Зашифровано(Одноразовый блокнот Шеннона)")
    print(cryp_text)
    decryp_text = dec_sh(cryp_text, key) # вызываем функцию расшифровки
    print("Расшифровано(Одноразовый блокнот Шеннона)")
    print(decryp_text)
```

## Тестирование:

```
Выберете шифр, которым хотели бы зашифровать:
ШИФРЫ ОДНОЗНАЧНОЙ ЗАМЕНЫ
1 - АТБАШ
2 - Шифр Цезаря
3 - Квадрат Полибия
ШИФРЫ МНОГОЗНАЧНОЙ ЗАМЕНЫ
4 - Шифр Белазо
5 - Шифр Третьяка
6 - Шифр Вижнера
ШИФРЫ БЛОЧНОЙ ЗАМЕНЫ
7 - Шифр Плейфера
8 - Матричный шифр
ШИФР ПЕРЕСТАНОВКИ
9 - Шифр Вертикальной Перестановки
ШИФР ГАФИРОВАНИЯ
10 - Одноразовый блокнот Шеннона
ПОТОЧНЫЙ ШИФР
11 - A5/1
КОМБИНАЦИОННЫЙ ШИФР
12 - Кузнечик
АССИМЕТРИЧНЫЕ ШИФРЫ(ГЕНЕРАЦИЯ ШИФРОВОЙ ПОДПИСИ)
13 - RSA
14 - ElGamal
15 - Обмен ключами по алгоритму Diffie-Hellman
16
Генерируем ключ для шифрования
[15, 20, 16, 31, 5, 30, 5, 27, 3, 7, 25, 26, 10, 17, 25, 5, 3, 16, 22, 10, 3, 21, 15, 32, 30, 16, 15, 17, 12, 21, 26, 24, 31, 12, 31]
Зашифровано(Одноразовый блокнот Шеннона)
ъвооеийиузапгблпхгхсвмщдлзясгй
Расшифровано(Одноразовый блокнот Шеннона)
леопарднемокетизменитъсвоиклянтенк
```

Блокнот Шенон:

Ключ для шифрования:

15, 29, 16, 31, 5 и 7.9. (скариф. аут. бор.)

леопард не может изменить свои кесты.

Зашифр. первые пять букв:

$$A = 11$$

$$N = 15$$

$$E = 5$$

$$A = 0$$

$$O = 14$$

Числа 10 и 1 меньше, т.к.

индексация происходит с 0

$$(11 + 15) \bmod 32 = 26 = \text{Г}$$

$$(5 + 29) \bmod 32 = 2 = \text{В}$$

$$(14 + 16) \bmod 32 = 30 = \text{Ю}$$

$$(15 + 31) \bmod 32 = 14 = \text{О}$$

$$(0 + 5) \bmod 32 = 5 = \text{Е}$$

Работа с текстом не менее 1000 знаков:

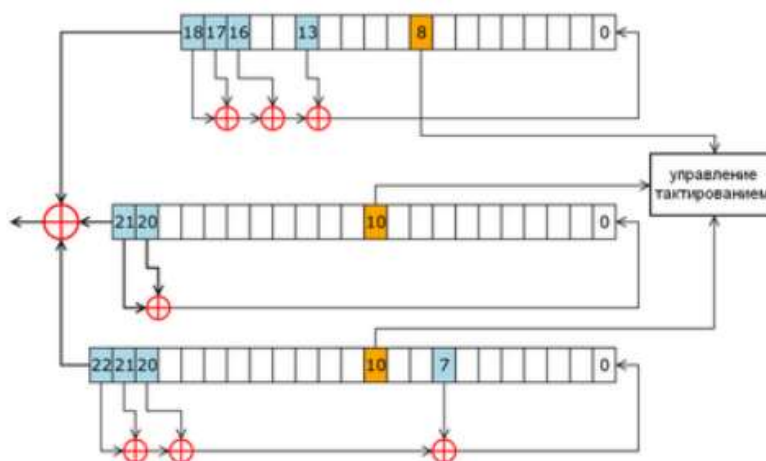
[illegible]

## Блок F: ПОТОЧНЫЕ ШИФРЫ

### 15. A5 /1

Алгоритм шифра, описание:

#### В АЛГОРИТМЕ ШИФРОВАНИЯ A5/1



Система РСЛОС в алгоритме A5/1:

Три регистра(R1, R2, R3) имеют длины 19, 22 и 23 бита,

Многочлены обратных связей:

$$X^{19} + X^{18} + X^{17} + X^{14} + 1 \text{ для R1}$$

$$X^{22} + X^{21} + 1 \text{ для R2}$$

$$X^{23} + X^{22} + X^{21} + X^8 + 1 \text{ для R3}$$

Управление тактированием:

биты синхронизации: 8 (R1), 10 (R2), 10 (R3)

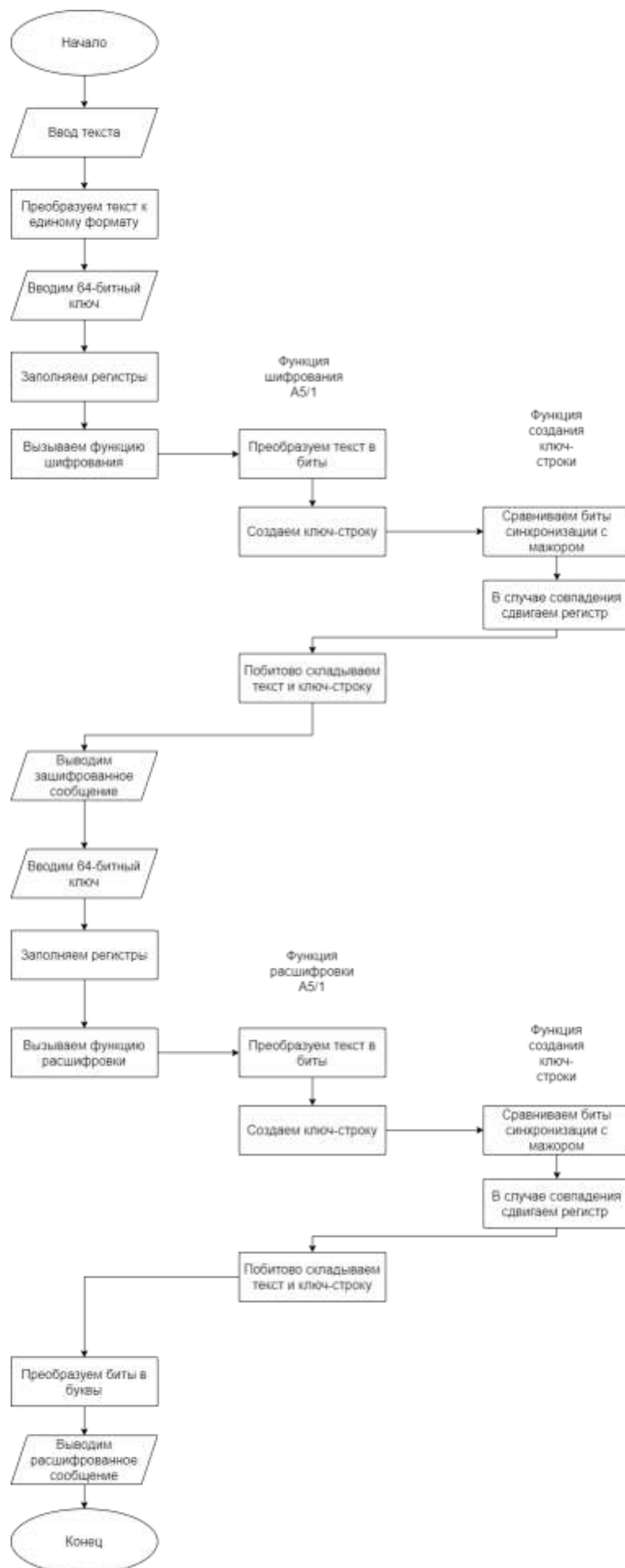
функция  $F = x \& y | x \& z | y \& z$ ,

где  $\&$  — булево AND,  $|$  - булево OR, x, y и z — биты синхронизации R1, R2 и R3

- сдвигаются только те регистры, у которых бит синхронизации равен F.

Выходной бит системы — результат операции XOR над выходными битами регистров.

Блок-схема программы:



## Код программы с комментариями:

```
import re
import copy

# Постоянные переменные
alph = "абвгдежзийклмнопрстуфхцщъыьэюя"
r1_length = 19
r2_length = 22
r3_length = 23
key_one = ""
r1 = []
r2 = []
r3 = []

def bin_enc(pr): #преобразуем строку в биты
    result = bin(int.from_bytes(pr.encode(encoding='utf-8'), 'big'))[2:]
    return result

def bin_dec(pr): # преобразуем биты в строку
    pr = int(pr, 2)
    result = pr.to_bytes((pr.bit_length() + 7) // 8, 'big').decode(encoding='utf-8')
    return result

def set_key(key): #записываем ключ в постоянные переменные
    key_one = key
    loading_registers(key) # вызываем функцию для заполнения регистров

def loading_registers(key): # функция заполнения регистров
    i = 0
    while(i < r1_length):
        r1.insert(i, int(key[i]))
        i = i + 1
    j = 0
    p = r1_length
    while(j < r2_length):
        r2.insert(j,int(key[p]))
        p = p + 1
        j = j + 1
    k = r2_length + r1_length
    r = 0
    while(r < r3_length):
        r3.insert(r,int(key[k]))
        k = k + 1
        r = r + 1

def get_majority(x,y,z): # функция F из методички, иначе говоря определяем число для
#сравнения с битом синхронизации
    if(x + y + z > 1):
        return 1
    else:
        return 0

def get_keystream(length): # функция изменения регистров
    r1_temp = copy.deepcopy(r1)
    r2_temp = copy.deepcopy(r2)
    r3_temp = copy.deepcopy(r3)
    keystream = []
```

```

i = 0
while i < length:
    majority = get_majority(r1_temp[8], r2_temp[10], r3_temp[10]) # вычисляем мажор
    if r1_temp[8] == majority: # если бит синхронизации совпадает с мажором, то меняем
регистр
        new = r1_temp[13] ^ r1_temp[16] ^ r1_temp[17] ^ r1_temp[18]
        r1_temp_two = copy.deepcopy(r1_temp)
        j = 1
        while(j < len(r1_temp)):
            r1_temp[j] = r1_temp_two[j-1]
            j = j + 1
        r1_temp[0] = new

    if r2_temp[10] == majority: # если бит синхронизации совпадает с мажором, то меняем
регистр
        new_one = r2_temp[20] ^ r2_temp[21]
        r2_temp_two = copy.deepcopy(r2_temp)
        k = 1
        while(k < len(r2_temp)):
            r2_temp[k] = r2_temp_two[k-1]
            k = k + 1
        r2_temp[0] = new_one

    if r3_temp[10] == majority: # если бит синхронизации совпадает с мажором, то меняем
регистр
        new_two = r3_temp[7] ^ r3_temp[20] ^ r3_temp[21] ^ r3_temp[22]
        r3_temp_two = copy.deepcopy(r3_temp)
        m = 1
        while(m < len(r3_temp)):
            r3_temp[m] = r3_temp_two[m-1]
            m = m + 1
        r3_temp[0] = new_two

    keystream.insert(i, r1_temp[18] ^ r2_temp[21] ^ r3_temp[22])
    i = i + 1
return keystream

def cryp_a5_1(pr):
    cryp_text = ""
    binary = bin_enc(pr)
    keystream = get_keystream(len(binary))
    i = 0
    while(i < len(binary)):
        cryp_text = cryp_text + str(int(binary[i]) ^ keystream[i])
        i = i + 1
    return cryp_text

def dec_a5_1(pr):
    decryp_text = ""
    binary = []
    keystream = get_keystream(len(pr))
    i = 0
    while(i < len(pr)):
        binary.insert(i, int(pr[i]))
        decryp_text = decryp_text + str(binary[i] ^ keystream[i])
        i = i + 1
    return bin_dec(decryp_text)

```

```
key = str(input("Введите 64-битный ключ: "))
print(key)
set_key(key)
cryp_text = cryp_a5_1(proverb)
print("Зашифровано(A5/1)")
print(cryp_text)
decryp_text = dec_a5_1(cryp_text)
print("Расшифровано(A5/1)")
print(decryp_text)
```

```
# Ключ: 0101001000011010110001110001100100101001000000110111111010110111
```

### Тестирование:

[illegible]



Шифр А5/1  
Леонард не может изменить  
свою метку.

Проверим первые три бита:  
Включено

Буква  $A = 11010000\ 1011\ 1011$

Ключ:  $0101\ 0010\ 0001\ 1010\ 1100$   
 $0111\ 0001\ 1001\ 0010\ 1001\ 0000$   
 $0011\ 0111\ 1110\ 1011\ 0111$

$R_1 = 0101\ 0010\ 0001\ 1010\ 1100$

$R_2 = 0\ 0111\ 0001\ 1001\ 0010\ 1001\ 0$

$R_3 = 000\ 0011\ 0111\ 1110\ 1011\ 0111$

1 Проверим бит анкр:

$f = 0$ , анкр. сбрасывается

$R_1$  и  $R_2$

Новый элемент для  $R_1$ :

$0 + 1 + 1 + 0 = 1$

$R_1 = 1010\ 1001\ 0000\ 1101\ 0111$

Наход 2-й для R2:

$$0 + 1 + 1 = 0$$

$$R_2 = 0001 \quad 1100 \quad 0110 \quad 0100 \quad 1010 \quad 01$$

Получили первое ключевое  
значение для сравнения:

$$1 + 1 + 1 = 0$$

Шифруем первый байт

$$0 + 1 = 0$$

Продолжаем аналогичные  
действия для остальных байтов:

0100 1000 0011 0001 - зашифр.  
сообщение:

Работа с текстом не менее 1000 знаков:



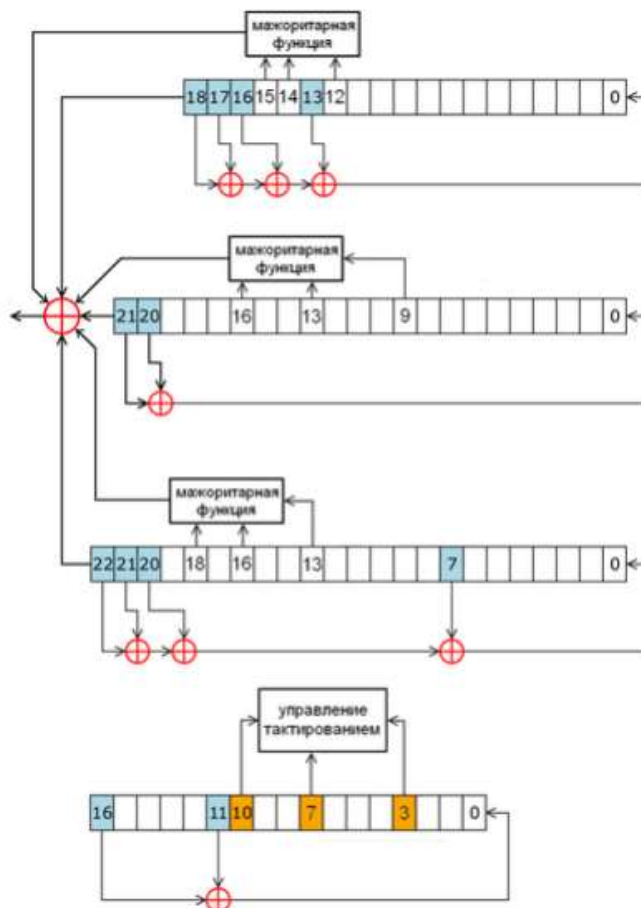


## Блок F: ПОТОЧНЫЕ ШИФРЫ

### 16. A5 /2

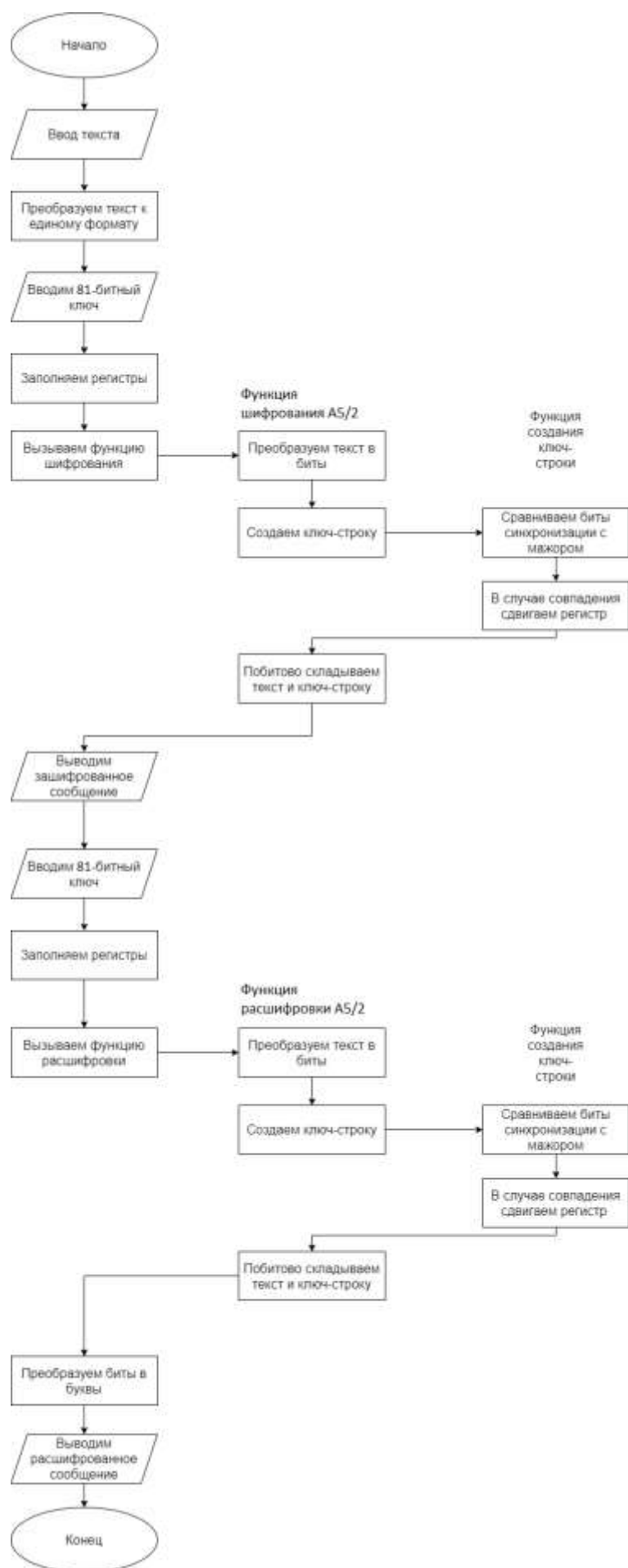
Алгоритм шифра, описание:

#### АЛГОРИТМ ШИФРОВАНИЯ A5/2



- добавлен регистр R4 длиной 17 бит

Блок-схема программы:



## Код программы с комментариями:

```
import re
import copy

# Постоянные переменные
alph = "абвгдежзийклмнопрстуфхцщщъыьэюя"
r1_length = 19
r2_length = 22
r3_length = 23
r4_length = 17
key_one = ""
r1 = []
r2 = []
r3 = []
r4 = []

def bin_enc(pr): #преобразуем строку в биты
    result = bin(int.from_bytes(pr.encode(encoding='utf-8'), 'big'))[2:]
    return result

def bin_dec(pr): # преобразуем биты в строку
    pr = int(pr, 2)
    result = pr.to_bytes((pr.bit_length() + 7) // 8, 'big').decode(encoding='utf-8')
    return result

def set_key(key): #записываем ключ в постоянные переменные
    key_one = key
    loading_registers(key) # вызываем функцию для заполнения регистров

def loading_registers(key): # функция заполнения регистров
    i = 0
    while(i < r1_length):
        r1.insert(i, int(key[i]))
        i = i + 1
    j = 0
    p = r1_length
    while(j < r2_length):
        r2.insert(j,int(key[p]))
        p = p + 1
        j = j + 1
    k = r2_length + r1_length
    r = 0
    while(r < r3_length):
        r3.insert(r,int(key[k]))
        k = k + 1
        r = r + 1
    y = 0
    m = r1_length + r2_length + r3_length
    while(y < r4_length):
        r4.insert(y,int(key[m]))
        m = m + 1
        y = y + 1

def get_majority(x,y,z): # функция F из методички, иначе говоря определяем число для
    #сравнения с битом синхронизации
    if(x + y + z > 1):
        return 1
```

```

else:
    return 0

def get_keystream(length): # функция изменения регистров
    r1_temp = copy.deepcopy(r1)
    r2_temp = copy.deepcopy(r2)
    r3_temp = copy.deepcopy(r3)
    r4_temp = copy.deepcopy(r4)
    keystream = []
    i = 0
    while i < length:
        majority = get_majority(r4_temp[3], r4_temp[7], r4_temp[10]) # вычисляем мажор
        if r4_temp[10] == majority: # если бит синхронизации совпадает с мажором, то меняем
регистр
            new = r1_temp[13] ^ r1_temp[16] ^ r1_temp[17] ^ r1_temp[18]
            r1_temp_two = copy.deepcopy(r1_temp)
            j = 1
            while(j < len(r1_temp)):
                r1_temp[j] = r1_temp_two[j-1]
                j = j + 1
            r1_temp[0] = new

            if r4_temp[3] == majority: # если бит синхронизации совпадает с мажором, то меняем
регистр
                new_one = r2_temp[20] ^ r2_temp[21]
                r2_temp_two = copy.deepcopy(r2_temp)
                k = 1
                while(k < len(r2_temp)):
                    r2_temp[k] = r2_temp_two[k-1]
                    k = k + 1
                r2_temp[0] = new_one

                if r4_temp[7] == majority: # если бит синхронизации совпадает с мажором, то меняем
регистр
                    new_two = r3_temp[7] ^ r3_temp[20] ^ r3_temp[21] ^ r3_temp[22]
                    r3_temp_two = copy.deepcopy(r3_temp)
                    m = 1
                    while(m < len(r3_temp)):
                        r3_temp[m] = r3_temp_two[m-1]
                        m = m + 1
                    r3_temp[0] = new_two

                keystream.insert(i, r1_temp[18] ^ r2_temp[21] ^ r3_temp[22])
                i = i + 1
    return keystream

def cryp_a5_2(pr):
    cryp_text = ""
    binary = bin_enc(pr)
    keystream = get_keystream(len(binary))
    i = 0
    while(i < len(binary)):
        cryp_text = cryp_text + str(int(binary[i]) ^ keystream[i])
        i = i + 1
    return cryp_text

def dec_a5_2(pr):
    decryp_text = ""
    binary = []
    keystream = get_keystream(len(pr))

```

```

i = 0
while(i < len(pr)):
    binary.insert(i,int(pr[i]))
    decrypt_text = decrypt_text + str(binary[i] ^ keystream[i])
    i = i + 1
return bin_dec(decrypt_text)

proverb = input("Введите текст для шифрования: ")
# proverb = "Леопард не может изменить своих пятен."
proverb = proverb.replace(" ", "")
proverb = proverb.lower()
while proverb.find(",") != -1:
    str1 = proverb[:proverb.find(",")]
    str2 = proverb[proverb.find(",")+1:]
    proverb = str1 + "зпт" + str2
while proverb.find(".") != -1:
    str1 = proverb[:proverb.find(".")]
    str2 = proverb[proverb.find(".")+1:]
    proverb = str1 + "тчк" + str2

key = str(input("Введите 64 + 17 ключ: "))
print(key)
set_key(key)
cryp_text = cryp_a5_2(proverb)
print("Зашифровано(A5/2)")
print(cryp_text)
decrypt_text = dec_a5_2(cryp_text)
print("Расшифровано(A5/2)")
print(decrypt_text)

# Ключ: 010100100001101011000111000110010010100100000011011111101011011101100001010011111

```

## Тестирование:

```

Введите текст для шифрования: Леопард не может изменить своих пятен.
Введите 64 + 17 ключ: 010100100001101011000111000110010010100100000011011111101011011101100001010011111
010100100001101011000111000110010010100100000011011111101011011101100001010011111
Зашифровано(A5/2)
01101101110011100101010101101100101000001011100101000100000110100110110111010100001000000111010011010011111001000110011011010101010101101011000001100001010010101001010101111110000000011
011100010101010100000111110100000001011000111011100011000001011001110000101010110101101011000001100001010010101001010101111110000000011
0000111000100101010001010011000101101001100001101000101100110100000101011001101010111110100010011110100010011100100011001111110010001001100
00101110001010111111010111101110010010001010111110101111110100111100101011001011001011001011001111110111
Расшифровано(A5/2)
Леопард не может изменить своих пятентчк

```



A 5/2 Умножение.

Ключ: 0101 0010 0001 1010  
 1100 0111 0001 1001 0010  
 1001 0000 0011 0111 1110 1011  
 0111 0110 0001 0100 1111 1

$R_1 = 0101 \quad 0010 \quad 0001 \quad 1010 \quad 110$

$R_2 = 0 \quad 0111 \quad 0001 \quad 1001 \quad 0010$   
 $1001 \quad 0$

$R_3 = 000 \quad 0011 \quad 0111 \quad 1110 \quad 100$   
 $1011 \quad 0111$

$R_4 = 0110^3 \quad 0001^{4567} \quad 0100^{8910} \quad 1111$

Проверка

$$f = 011/010/110 = 0$$

то дат смкр = 0, еще  $R_1$   
 едновременно:

Коды 21-7 гмс  $R_1$

$$0 + 1 + 1 + 0 = 1$$

$R_1 = 1010 \quad 1001 \quad 0000 \quad 1101 \quad 011$

3 бита шифр = 0, след.  
 R 2 удваивается:  
 $0 + 1 + 1 = 0$   
 $R_2 = 0001 \quad 1100 \quad \cancel{00110} \quad 0100$   
 $1010 \quad 01$   
 Первое кин. знач.:  
 $1 + 1 + 1 = 0$   
 Шифр. продв. бит:  
 $0 + 1 = 0$   
 Прове. аналог. для перв.  
 буквы:  
 $0110 \quad 1101 \quad 1100 \quad 1110$

Работа с текстом не менее 1000 знаков:

всёте текст для зашифровки: Людина, що була руса, Будильник, Мил, да був будильник. У него були уся, шипа і серце. І он не хотів кінтяться. Он реалі кінтяться, когда стукает без пятадцати девять. Ровно в восемь он сделал предложение графину с водор. Графин с водор согласился немедленно, но в пятадцати минут девятьго его унесли и выдали замук за водопроводчый крак. Дело было сделано, и Графин вернуся на стол к Будильнику уже замужней дамой. Было двадцать минут девятьго. Бремене оставалось мало. Будильник тогда сделал предложение очкам. Очки были старые и неоднократно вылодили замук за уся. Очки подумали пять минут и согласились, но в этот момент их опять выдали замук за уся. Было уже восемь часов двадцать пять минут. Тогда будильник быстро сделал предложение книге. Книга тут же согласилась, и будильник стал читать, когда же стукает без пятадцати девять. Сердце его очень громко колотилось. Тут его взяли и надели посуду, потому что детей уложили спать. И без пятадцати девять будильник неожиданно для себя женился на посуде.

Будите 64 + 17 ключ: 01010010000110101100011100011001001010010000001101111110101101110110000101001111

010100100001101011000111000110010010100100000011011111101011011101100001010011111

Задача 45/2)

[illegible][illegible]

Расшифрование (45/2)

[illegible]

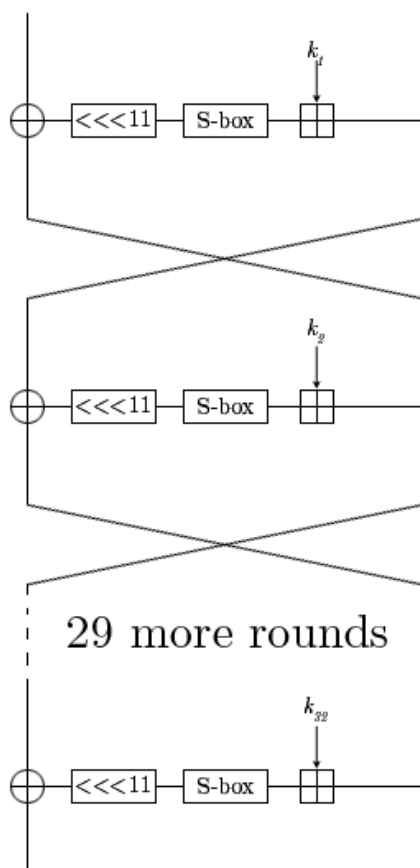


## Блок G: КОМБИНАЦИОННЫЕ ШИФРЫ

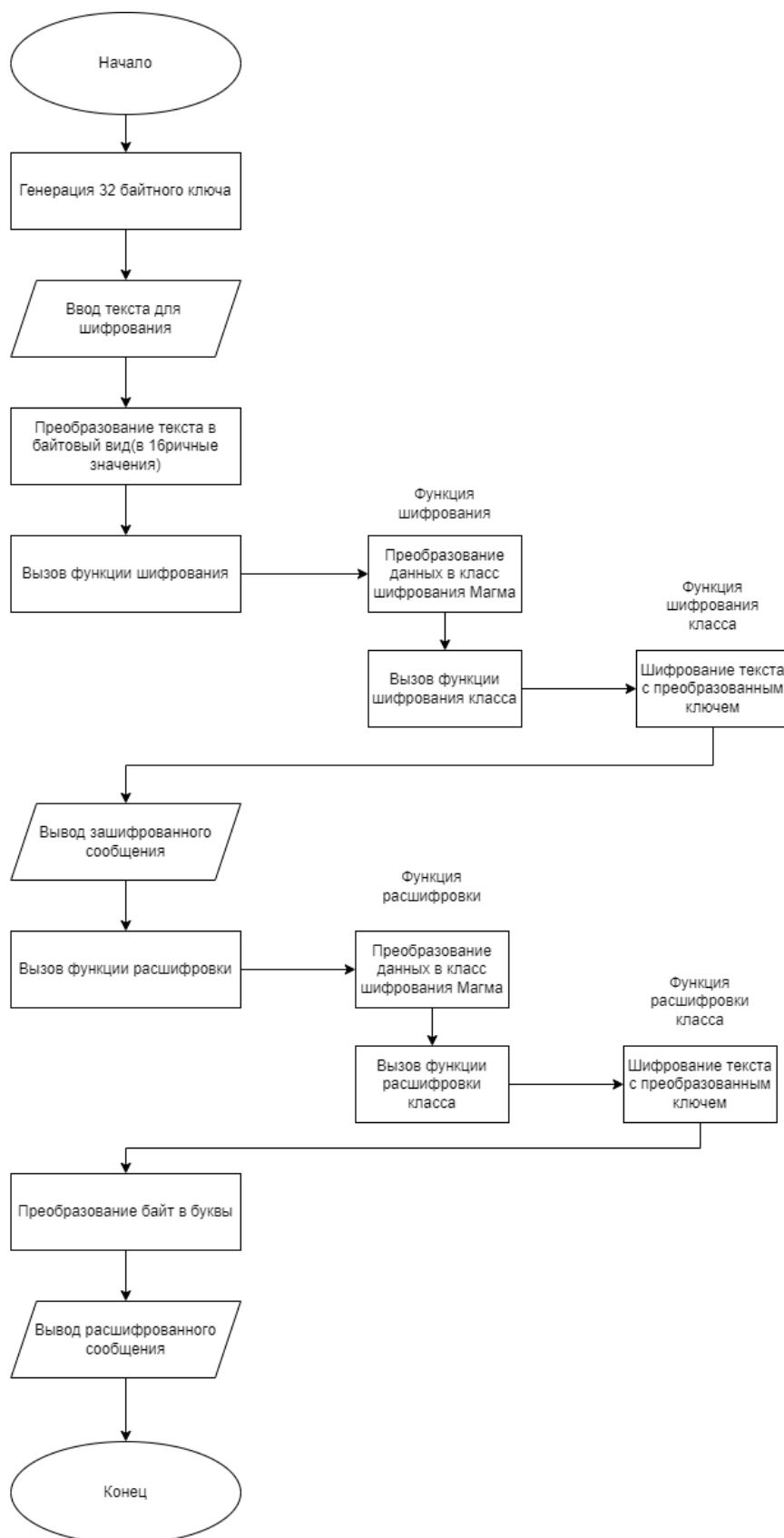
### 17.МАГМА

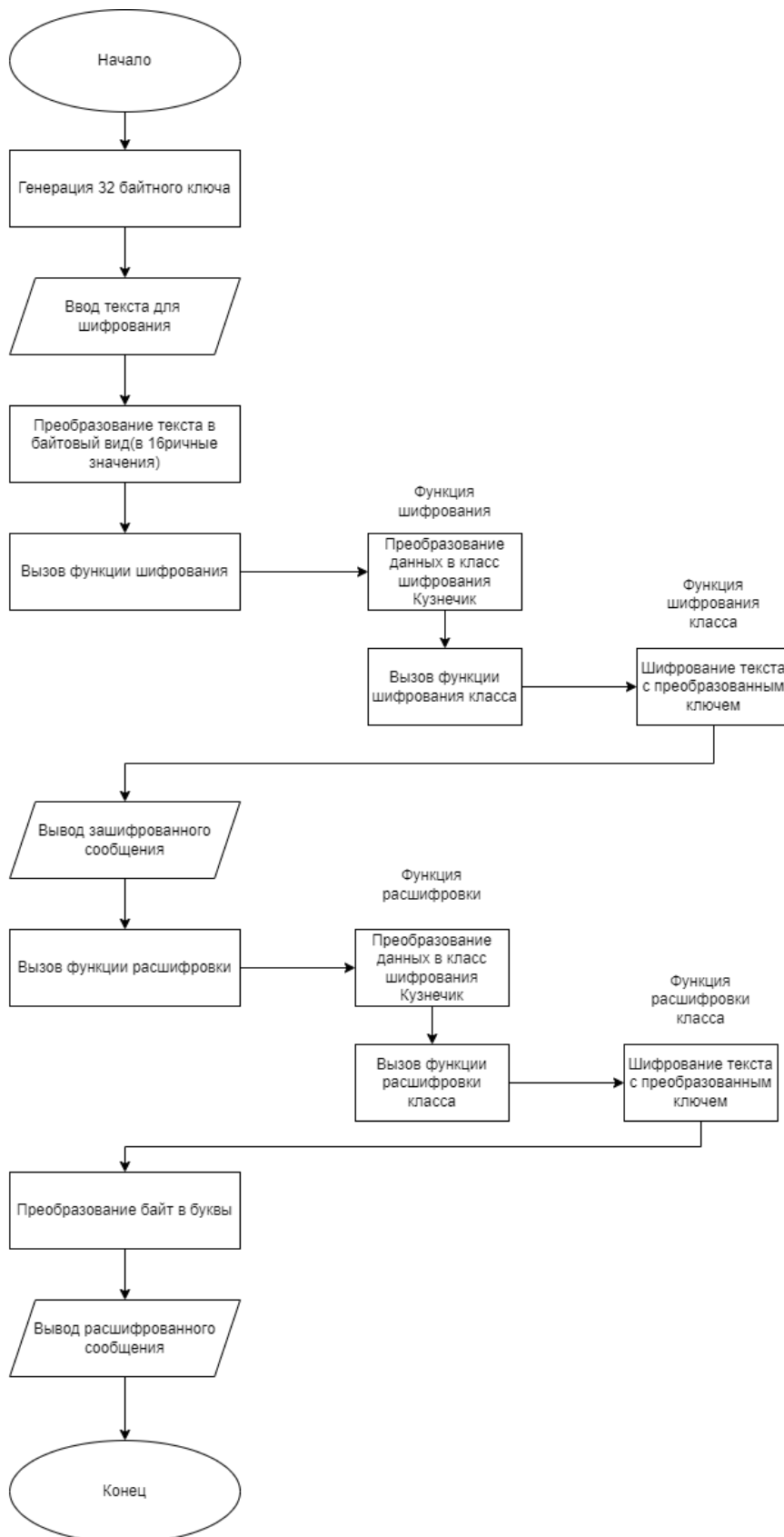
Алгоритм шифра, описание:

Магма – это шифр, который определен в ГОСТ Р 34.12-2015, по факту тот же самый ГОСТ 28147 89, только в профиль. Шифруемый блок проходит 32 раунда, в которых происходят некоторые преобразования. Ключ состоит из 256 бит, каждый раундовый ключ представляет собой часть исходного.



Блок-схема программы:





Код программы с комментариями:

```

# -*- coding: utf-8

import sys
from math import ceil

```

```

from pygost.gost3412 import GOST3412Magma
from pygost.utils import hexdec, hexenc
from os import urandom

def gost28147_ns2block(ns): # сеть Фейстеля для шифрования

    n1, n2 = ns
    return bytes(bytearray((
        (n2 >> 0) & 255, (n2 >> 8) & 255, (n2 >> 16) & 255, (n2 >> 24) & 255,
        (n1 >> 0) & 255, (n1 >> 8) & 255, (n1 >> 16) & 255, (n1 >> 24) & 255,
    )))

def gost28147_encrypt(sbox, key, ns): # поблочное шифрование
    return xcrypt(SEQ_ENCRYPT, sbox, key, ns)

def gost28147_decrypt(sbox, key, ns): #поблочная расшифровка
    return xcrypt(SEQ_DECRYPT, sbox, key, ns)

def gost28147_block2ns(data): # сеть Фейстеля для расшифровки
    data = bytearray(data)
    return (
        data[0] | data[1] << 8 | data[2] << 16 | data[3] << 24,
        data[4] | data[5] << 8 | data[6] << 16 | data[7] << 24,
    )

def xcrypt(seq, sbox, key, ns): #сама функция шифрования/расшифровки
    s = SBOXES[sbox]
    w = bytearray(key)
    x = [
        w[0 + i * 4] |
        w[1 + i * 4] << 8 |
        w[2 + i * 4] << 16 |
        w[3 + i * 4] << 24 for i in range(8)
    ]
    n1, n2 = ns
    for i in seq:
        n1, n2 = _shift11(_K(s, addmod(n1, x[i]))) ^ n2, n1
    return n1, n2

class GOST3412Magma(object): # класс шифрования Магма
    """GOST 34.12-2015 64-bit block cipher Магма (Magma)
    """
    def __init__(self, key):
        self.key = b"".join(key[i * 4:i * 4 + 4][::-1] for i in range(8))
        self.sbox = data

    def encrypt(self, blk): #шифрование Магма3412, которое базируется на ГОСТ 28147
        return gost28147_ns2block(gost28147_encrypt(
            self.sbox,
            self.key,
            gost28147_block2ns(blk[::-1]),
        ))[::-1]

    def decrypt(self, blk): #расшифровка Магма3412, которое базируется на ГОСТ 28147
        return gost28147_ns2block(gost28147_decrypt(
            self.sbox,
            self.key,
            gost28147_block2ns(blk[::-1]),
        ))[::-1]

```

```

flatten = lambda l: [item for sublist in l for item in sublist]

def to_chunks(lst, n = 8):
    lst += bytes([0]) * (ceil(len(lst) / n) * n - len(lst))
    return [lst[i:i + n] for i in range(0, len(lst), n)]

key = None

def set_key(_key):
    global key
    key = hexdec(_key)

def encrypt_magma(data : bytearray):
    global key
    ciph = GOST3412Magma(key) # задаем переменную класса шифрования магма
    return bytearray(flatten([ciph.encrypt(chunk) for chunk in to_chunks(data, 8)])) #
    шифруем, вызывая функции из класса

def decrypt_magma(data : bytearray):
    global key
    ciph = GOST3412Magma(key) # задаем переменную класса шифрования кузнечика
    return bytearray(flatten([ciph.decrypt(chunk) for chunk in to_chunks(data, 8)])) #
    расшифровываем, вызывая функции из класса

def main():
    key = hexenc(urandom(32)) #Генерация 32битного ключа
    set_key(key)
    # pr = "Леопард не может изменить своих пятен."
    # pr = " Людмила Петрушевская. Будильник. Жил, да был будильник. У него были усы, шляпа
    и сердце. И он решил жениться. Он решил жениться, когда стукнет без пятнадцати девять.
    Ровно в восемь он сделал предложение графину с водой. Графин с водой согласился немедленно,
    но в пятнадцать минут девятого его унесли и выдали замуж за водопроводный кран. Дело было
    сделано, и графин вернулся на стол к будильнику уже замужней дамой. Было двадцать минут
    девятого. Времени оставалось мало. Будильник тогда сделал предложение очкам. Очки были
    старые и неоднократно выходили замуж за уши. Очки подумали пять минут и согласились, но в
    этот момент их опять выдали замуж за уши. Было уже восемь часов двадцать пять минут. Тогда
    будильник быстро сделал предложение книге. Книга тут же согласилась, и будильник стал
    ждать, когда же стукнет без пятнадцати девять. Сердце его очень громко колотилось. Тут его
    взяли и накрыли подушкой, потому что детей уложили спать. И без пятнадцати девять будильник
    неожиданно для себя женился на подушке."
    pr = input("Введите текст: ")
    data = pr.encode('utf-8')
    print(' '.join([str(e) for e in encrypt_magma(data)]))
    cryp_text = ' '.join([str(e) for e in encrypt_magma(data)])

    data = bytes(int(s) for s in cryp_text.split(' '))
    print(decrypt_magma(data).decode('utf-8'))

if __name__ == "__main__":
    main()

```



Программа для тестирования(используется готовая библиотека, функции которой подробно описаны выше):

```
from pygost.gost3412 import GOST3412Kuznechik
from pygost.gost3412 import GOST3412Magma
from pygost.utils import hexdec, hexenc

key = hexdec("ffeeddccbbaa99887766554433221100f0f1f2f3f4f5f6f7f8f9fafbfcfdfeff")
plaintext = hexdec("fedcba9876543210")
ciphertext = hexdec("4ee901e5c2d8ca3d")

ciph = GOST3412Magma(key)
print("Ключ: ", hexenc(key))
print("Обычный текст: ", hexenc(plaintext))
print("Зашифрованный текст:", hexenc(ciph.encrypt(plaintext)))
print("Зашифрованный текст для проверки: ", hexenc(ciphertext))
print()
```

Тестирование:

Тестирование по ГОСТ:

```
Ключ: ffeeddccbbaa99887766554433221100f0f1f2f3f4f5f6f7f8f9fafbfcfdfeff
Обычный текст: fedcba9876543210
Зашифрованный текст: 4ee901e5c2d8ca3d
Зашифрованный текст для проверки: 4ee901e5c2d8ca3d
```

```
215 36 198 73 129 18 42 213 177 14 198 7 184 190 134 131 5 158 149 42 1 144 188 157 23 173 115 15 192 110 211 223 130 238 45 33 154 162 10 105 56 135 19
5 131 35 113 207 237 178 185 29 68 91 120 29 59 93 161 120 0 35 159 234 17 62 83 141 125 160 207 153
Леопард не может изменить своих пятен.
```

Работа с текстом не менее 1000 знаков:

```
0 111 151 4 255 143 41 183 127 48 69 185 163 216 117 42 21 52 225 191 181 91 252 200 168 73 67 34 221 236 175 26 152 118 72 197 89 78 127 72 73 113 135
250 93 97 187 239 213 72 55 79 155 34 236 222 233 210 134 62 189 172 235 127 176 180 98 57 221 190 1 207 67 165 226 39 157 157 211 59 121 75 163 38 35 1
74 170 249 121 88 22 183 126 160 163 174 122 231 63 162 54 25 187 24 189 194 55 120 225 104 242 138 56 207 118 132 98 178 190 183 67 163 141 91 58 83 22
8 145 232 8 112 41 141 88 253 152 19 169 7 213 207 121 42 55 237 81 9 174 51 66 97 6 39 128 228 248 10 5 40 78 110 224 70 225 116 23 116 38 71 175 161 1
90 61 226 137 201 103 17 135 171 184 45 206 73 231 84 133 85 71 178 26 185 88 193 239 209 86 40 218 200 56 84 148 136 36 40 190 159 197 215 100 66 193 1
28 145 68 184 4 146 94 243 0 64 132 218 203 223 111 175 42 191 224 119 208 227 124 179 35 35 12 217 130 18 146 192 161 133 187 17 204 42 168 121 51 65 7
5 32 205 229 145 187 242 81 144 69 92 88 114 249 53 116 130 75 50 125 64 34 187 38 243 50 184 37 146 70 60 200 94 241 82 151 250 177 122 64 19 4 236 97
98 208 245 123 226 162 8 121 27 141 126 38 205 236 201 73 176 55 49 68 0 121 181 120 217 180 202 142 6 224 131 163 32 23 184 26 217 182 123 168 189 161
263 78 138 75 141 11 62 189 172 235 127 170 188 98 216 238 70 45 194 217 186 245 194 24 77 38 155 240 183 197 121 8 233 61 169 227 133 188 208 26 190 22
224 40 251 30 240 163 93 59 169 72 187 117 140 70 181 156 24 16 232 254 121 32 114 7 212 86 24 249 137 156 33 111 139 75 24 105 128 101 5 146 87 77 150
119 181 93 203 228 87 236 247 48 112 38 228 151 161 187 187 211 86 153 135 195 224 17 134 245 27 144 247 165 223 85 43 162 40 229 231 175 33 195 252 12
5 254 55 22 246 17 49 130 151 55 133 4 29 79 113 174 182 70 0 250 229 110 194 136 136 231 47 12 149 37 238 36 131 156 37 124 137 50 203 241 123 99 179 9
5 163 6 177 174 28 20 159 124 165 3 47 118 116 213 8 175 174 128 25 41 147 125 118 232 215 35 201 225 9 185 183 191 131 197 134 155 184 51 254 195 82 19
9 158 88 232 245 223 116 174 71 9 138 235 81 135 61 144 214 61 169 57 154 190 95 214 74 252 151 56 53 177 8 125 158 53 90 204 85 129 247 45 95 4 181 126
206 99 6 155 155 124 115 6 75 131 82 183 135 126 214 26 240 244 105 183 242 196 243 180 68 168 217 215 84 236 175 192 14 36 165 221 149 122 171 201 49
216 187 156 124 116 12 138 103 119 62 9 213 169 244 28 173 218 78 86 209 178 162 80 158 238 68 28 21 122 136 55 222 132 200 32 204 228 63 110 21 200 219
119 191 130 231 35 196 170 39 128 124 12 206 38 229 242 71 177 46 15 41 32 206 147 98 118 43 146 129 159 22 2 23 173 171 183 53 7 255 153 224 103 151 1
01 158 118 133 174 112 114 113 195 24 254 201 215 145 237 250 50 95 18 193 55 24 130 172 182 135 70 58 172 21 13 188 219 95 91 90 72 228 210 71 205 123
187 167 151 118 22 29 24 170 198 121 152 142 162 137 106 1 133 24 56 148 55 156 160 226 78 81 99 112 125 41 162 154 122 97 42 199 80 150 114 45 208 140
126 224 31 14 117 73 88 238 39 101 101 157 182 253 10 201 162 88 221 156 181 47 107 167 55 86 225 247 253 41 132 66 82 98 135 90 224 37 247 0 25 175 204
194 161 10 7 187 225 65 144 220 118 8 128 68 105 201 31 207 45 57 250 117 87 83 222 247 14 182 167 215 229 37 40 61 143 131 244 0 194 210 204 240 50 46
36 213 183 164 28 26 209 170 243 171 198 136 40 63 240 83 185 152 173 12 193 185 181 158 239 67 167 2 223 174 153 144 222 45 164 120 170 135 111 69 244
198 148 75 17 124 127 97 8 232 29 112 255 245 108 115 213 26 141 188 86 200 208 152 181 22 1 218 84 35 114 152 129 25 91 241 125 27 167 219 33 219 71 2
42 75 16 195 48 141 247 124 58 162 2 206 29 79 114 49 47 223 32 245 211 88 60 154 34 0 174 194 246 41 132 244 183 189 158 166 188 8 24 171 143 150 42 13
7 50 169 41 120 99 201 38 156 60 202 6 71 73 110 9 236 180 116 81 135 61 144 214 61 160 57 154 198 95 214 74 252 151 56 190 90 240 34 125 26 226 219 73
145 235 147 234 216 9 29 125 106 90 164 30 142 187 41 18 112 148 128 25 37 171 205 77 129 66 127 75 20 45 22 27 132 222 125 74 186 245 124 126 6 136 144
206 12 215 240 217 180 202 142 6 224 131 163 176 238 160 88 183 126 51 182
```

Лидмила Петрушевская. Будильник. Жил, да был будильник. У него были усы, шипы и сердце. И он решил жениться. Он решил жениться, когда стукнет без пяти аддати девять. Ровно в восемь он сделал предложение графине с водой. Графин с водой согласился немедленно, но в пятнадцать минут девятого его унесли и в ыдали замуж за водопроводный кран. Дело было сделано, и графин вернулся на стол к будильнику уже замужней дамой. Было двадцать минут девятого. Времени о ставилось мало. Будильник тогда сделал предложение очкам. Очки были старые и неоднократно выкидывали замуж за уши. Очки подумали пять минут и согласились, но в этот момент их опыты выдали замуж за уши. Было уже восемь часов двадцать пять минут. Тогда будильник быстро сделал предложение книге. Книга тут же согласилась, и будильник стал ждать, когда же стукнет без пятнадцати девять. Сердце его очень громко колотилось. Тут его взяли и накрыли подушкой, пото му что детей уложили спать. И без пятнадцати девять будильник неожиданно для себя женился на подушке.

## Блок G: КОМБИНАЦИОННЫЕ ШИФРЫ

### 20.КУЗНЕЧИК

Алгоритм шифра, описание:

#### Модель для шифра ГОСТ Р 34.12-2015 «Кузнечик»

Шифр принадлежит к классу LSX-шифров: его базовое преобразование (функция шифрования блока) представляется десятью циклами последовательных преобразований  $L$  (линейное преобразование),  $S$  (подстановка) и  $X$  (смешивание с цикловым ключом) [3].

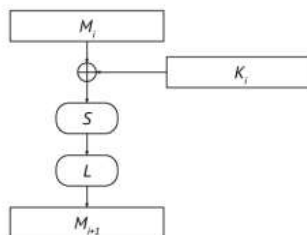


Рисунок 14 — Полный цикл базового преобразования

Блок-схема программы:

Код программы с комментариями:

```
# -*- coding: utf-8
import sys
from math import ceil
from pygost.gost3412 import GOST3412Kuznechik
from pygost.utils import hexdec, hexenc
from os import urandom

def Linv(blk): # линейное преобразование
    for _ in range(16):
        t = blk[0]
        for i in range(15):
            blk[i] = blk[i + 1]
            t ^= GF[blk[i]][LC[i]]
        blk[15] = t
    return blk

def strxor(a, b): # Эта функция будет обрабатывать только самую короткую длину обеих строк,
# игнорируя оставшуюся.
    mlen = min(len(a), len(b))
    a, b, xor = bytearray(a), bytearray(b), bytearray(mlen)
    for i in xrange(mlen):
        xor[i] = a[i] ^ b[i]
    return bytes(xor)

def lp(blk):
    return L([PI[v] for v in blk])

class GOST3412Kuznechik(object): # класс шифрования
    """GOST 34.12-2015 128-bit block cipher Кузнечик (Kuznechik)
```

```

"""
def __init__(self, key):
    """
    :param key: encryption/decryption key
    :type key: bytes, 32 bytes
    Key scheduling (roundkeys precomputation) is performed here.
    """
    kr0 = bytearray(key[:16]) # разделяем ключ пополам
    kr1 = bytearray(key[16:]) # разделяем ключ пополам
    self.ks = [kr0, kr1]
    for i in range(4):
        for j in range(8):
            k = lp(bytearray(strxor(C[8 * i + j], kr0)))
            kr0, kr1 = [strxor(k, kr1), kr0]
        self.ks.append(kr0)
        self.ks.append(kr1)

def encrypt(self, blk): # функция шифрования по кузнечик
    blk = bytearray(blk)
    for i in range(9):
        blk = lp(bytearray(strxor(self.ks[i], blk)))
    return bytes(strxor(self.ks[9], blk))

def decrypt(self, blk): # функция расшифровки по кузнечик
    blk = bytearray(blk)
    for i in range(9, 0, -1):
        blk = [PIinv[v] for v in Linv(bytearray(strxor(self.ks[i], blk)))]
    return bytes(strxor(self.ks[0], blk))

flatten = lambda l: [item for sublist in l for item in sublist]

def to_chunks(lst, n = 8): # делим на куски по 8 бит
    lst += bytes([0]) * (ceil(len(lst) / n) * n - len(lst))
    return [lst[i:i + n] for i in range(0, len(lst), n)]

key = None

def set_key(_key):
    global key
    key = hexdec(_key)

def encrypt_kuz(data : bytearray):
    global key
    ciph = GOST3412Kuznechik(key) # задаем переменную класса шифрования кузнечика
    return bytearray(flatten([ciph.encrypt(chunk) for chunk in to_chunks(data, 16)])) #
шифруем, вызывая функции из класса

def decrypt_kuz(data : bytearray):
    global key
    ciph = GOST3412Kuznechik(key) # задаем переменную класса шифрования кузнечика
    return bytearray(flatten([ciph.decrypt(chunk) for chunk in to_chunks(data, 16)])) #
расшифровываем, вызывая функции из класса

def main():
    key = hexenc(urandom(32)) #Генерация 32битного ключа
    set_key(key)
    pr = "Леопард не может изменить своих пятен."

```

# pr = " Людмила Петрушевская. Будильник. Жил, да был будильник. У него были усы, шляпа и сердце. И он решил жениться. Он решил жениться, когда стукнет без пятнадцати девять. Ровно в восемь он сделал предложение графину с водой. Графин с водой согласился немедленно, но в пятнадцать минут девятого его унесли и выдали замуж за водопроводный кран. Дело было сделано, и графин вернулся на стол к будильнику уже замужней дамой. Было двадцать минут девятого. Времени оставалось мало. Будильник тогда сделал предложение очкам. Очки были старые и неоднократно выходили замуж за уши. Очки подумали пять минут и согласились, но в этот момент их опять выдали замуж за уши. Было уже восемь часов двадцать пять минут. Тогда будильник быстро сделал предложение книге. Книга тут же согласилась, и будильник стал ждать, когда же стукнет без пятнадцати девять. Сердце его очень громко колотилось. Тут его взяли и накрыли подушкой, потому что детей уложили спать. И без пятнадцати девять будильник неожиданно для себя женился на подушке."

```
# pr = input("Введите текст: ")
data = pr.encode('utf-8')
print(' '.join([str(e) for e in encrypt_kuz(data)]))
cryp_text = ' '.join([str(e) for e in encrypt_kuz(data)])
```

```
data = bytes(int(s) for s in cryp_text.split(' '))
print(decrypt_kuz(data).decode('utf-8'))
```

```
if __name__ == "__main__":
    main()
```

## Тестирование по ГОСТ:

```
Key: 8899aabbccddeeff0011223344556677fedcba98765432100123456789abcdef
Plain text: 1122334455667700ffeeddccbbaa9988
Encrypted text: 7f679d90bebc24305a468d42b9d4edcd
Expected: 7f679d90bebc24305a468d42b9d4edcd
```

## Работа с пословицей:

```
171 03 159 231 191 110 80 229 204 141 09 147 227 205 28 143 40 233 216 237 29 98 198 72 222 178 09 190 97 137 121 21 193 122 255 97 215 127 182 230 157
26 1 150 73 37 226 3 248 214 5 36 5 134 204 48 196 183 47 126 45 128 206 193 157 77 247 8 137 121 111 234 248 214 192 155 45 29 61 95
Дальше не имеет смысла изменять список.
```

## Работа с текстом не менее 1000 знаков:

```
4 173 19 135 105 130 170 86 47 203 252 10 47 90 213 184 08 165 38 143 89 41 212 140 210 81 70 133 238 191 140 149 122 80 141 27 180 81 214 185 10 134 12
0 155 163 08 189 03 14 242 11 90 174 48 202 195 22 83 156 187 99 40 50 138 224 156 192 241 230 154 138 132 96 103 112 130 226 136 232 148 140 152 190 94
14 190 143 201 36 08 117 35 212 215 227 130 215 210 204 74 28 50 80 97 125 135 137 135 47 242 177 160 110 187 14 231 12 182 111 108 202 41 130 214 61 4
5 111 134 187 77 182 171 144 58 05 34 15 156 80 215 184 78 119 216 163 247 88 77 136 233 60 253 45 138 39 207 184 243 41 244 21 225 228 251 7 124 217 6
214 02 108 58 170 189 81 38 82 128 197 51 29 230 3 180 184 190 26 148 142 135 128 17 67 50 40 148 93 2 192 161 189 209 117 82 70 58 145 243 33 224 165 5
118 185 128 177 132 111 161 18 217 62 13 142 06 242 159 197 221 136 48 242 208 247 38 62 171 122 122 55 181 244 136 205 32 78 124 119 62 198 140 135 11
1 143 190 23 248 107 26 248 34 194 252 207 210 160 1 51 221 162 171 170 159 130 89 203 208 111 8 146 53 131 33 56 162 246 172 181 93 32 82 112 130 158 1
75 73 244 240 18 05 211 88 242 126 221 13 50 123 107 16 191 120 235 206 161 171 180 90 108 04 225 22 229 243 177 187 232 220 242 154 85 07 40 91 244 181
125 54 232 79 52 110 77 135 163 240 31 234 242 216 121 188 21 77 174 67 205 211 150 36 174 153 113 215 186 13 212 158 203 216 20 21 171 127 196 08 255
9 173 221 56 4 50 220 145 194 210 110 142 165 194 134 10 118 178 197 160 164 67 187 29 41 87 188 148 215 8 190 75 170 17 150 54 172 9 243 184 98 202 17
7 52 235 170 238 0 219 197 21 83 53 194 232 129 206 173 95 163 20 36 241 156 53 65 124 210 171 43 77 164 73 120 120 210 248 163 184 153 75 195 162 190 9
8 239 2 75 212 45 212 87 12 99 219 227 198 118 0 129 118 134 140 207 03 193 02 46 175 191 192 126 7 92 172 148 225 131 107 78 188 190 188 147 105 137 17
4 186 16 18 143 51 55 176 64 210 5 35 186 226 232 97 88 44 81 206 74 137 186 222 212 76 211 123 135 165 237 4 182 72 123 182 125 28 170 59 118 237 94 2
252 123 152 302 262 83 2 222 92 80 221 15 170 27 59 132 181 169 216 183 41 180 142 193 125 157 118 255 108 238 230 105 247 170 184 160 111 100 108 161 7
8 214 118 34 38 80 234 237 213 18 241 122 214 23 31 145 94 73 46 183 163 124 133 56 121 162 162 184 181 56 186 124 40 83 168 174 162 203 45 190 129 10 5
4 143 63 135 12 160 159 18 193 179 172 38 204 46 136 47 185 37 223 28 158 177 135 54 9 187 235 112 122 205 176 232 170 90 188 103 244 242 143 167 85 72
220 81 54 104 09 201 119 103 120 37 234 52 21 59 46 180 83 106 133 130 97 233 190 238 196 70 187 217 18 06 238 149 192 70 48 72 3 214 241 234 161 170 14
1 138 144 96 231 193 48 147 88 79 195 242 77 32 35 251 65 14 231 34 153 138 30 212 240 129 241 238 241 235 94 33 41 7 77 73 179 30 238 126 113 258 242 1
57 242 81 145 133 78 154 187 30 253 125 131 136 252 156 177 118 187 41 82 162 86 186 140 29 36 161 15 32 121 184 174 214 65 55 166 122 123 156 25 2 82 14
0 258 223 72 54 51 0 29 91 254 170 60 94 134 168 246 77 94 22 150 224 118 244 143 57 36 14 108 88 10 61 165 113 182 18 60 227 166 236 10 235 144 59 128
94 243 230 252 40 168 162 184 67 248 44 211 184 128 207 68 189 33 190 235 165 23 210 4 72 21 150 226 40 156 130 28 191 56 247 219 245 208 29 113 153 58
18 246 183 167 284 96 241 227 248 134 25 123 231 52 113 4 11 134 123 129 18 242 384 178 248 26 72 139 67 47 170 168 14 162 53 132 220 190 188 226 152 14
7 162 1 64 31 221 37 199 0 177 254 1 11 150 125 258 178 58 2 47 245 232 87 78 31 215 74 204 1 188 189 220 190 128 52 16 168 30 79 174 168 186 180 138 98
190 91 53 133 245 236 28 83 22 58 4 175 34 86 123 54 216 90 77 12 116 129 78 189 215 45 232 14 167 247 226 113 236 3 89 92 57 223 228 156 38 132 182 64
186 31 138 98 48 147 178 36 172 46 23 138 104 224 84 235 144 179 185 137 65 5 205 233 180 135 183 44 213 187 165 96 129 88 6 7 255 93 67 183 78 29 168
218 115 111 189 203 247 187 151 234 203 160 235 113 115 73 7 171 200 4 175 97 66 68 186 84 236 184 16 181 64 0 166 6 212 140 151 222 26 240 137 19 135 1
```

Людмила Петрушевская. Будильник. Жил, да был будильник. У него были усы, шляпа и сердце. И он решил жениться. Он решил жениться, когда стукнет без пятнадцати девять. Ровно в восемь он сделал предложение графину с водой. Графин с водой согласился немедленно, но в пятнадцать минут девятого его унесли и выдали замуж за водопроводный кран. Дело было сделано, и графин вернулся на стол к будильнику уже замужней дамой. Было двадцать минут девятого. Времени оставалось мало. Будильник тогда сделал предложение очкам. Очки были старые и неоднократно выходили замуж за уши. Очки подумали пять минут и согласились, но в этот момент их опять выдали замуж за уши. Было уже восемь часов двадцать пять минут. Тогда будильник быстро сделал предложение книге. Книга тут же согласилась, и будильник стал ждать, когда же стукнет без пятнадцати девять. Сердце его очень громко колотилось. Тут его взяли и накрыли подушкой, потому что детей уложили спать. И без пятнадцати девять будильник неожиданно для себя женился на подушке.

## БЛОК Н: АСИММЕТРИЧНЫЕ ШИФРЫ

### 21. Шифр RSA

Алгоритм шифра, описание:

Берутся два очень больших простых числа  $P$  и  $Q$  и находится произведение простых чисел

$$N = P \times Q$$

и функция Эйлера от этого произведения

$$\phi(N) = (P-1) \times (Q-1).$$

Выбирается случайное целое число  $E$ , взаимно простое с  $\phi(N)$ , и вычисляется

$$D = (1 \bmod \phi(N)) / E.$$

$E$  и  $N$  публикуются как открытый ключ,  $D$  сохраняется в тайне.

*Шифрование:*

Если  $M$  — сообщение, то шифртекст  $C_i$  получается последовательным шифрованием каждой шифрвеличины  $M_i$  возведением ее в степень  $E$  по модулю  $N$ :

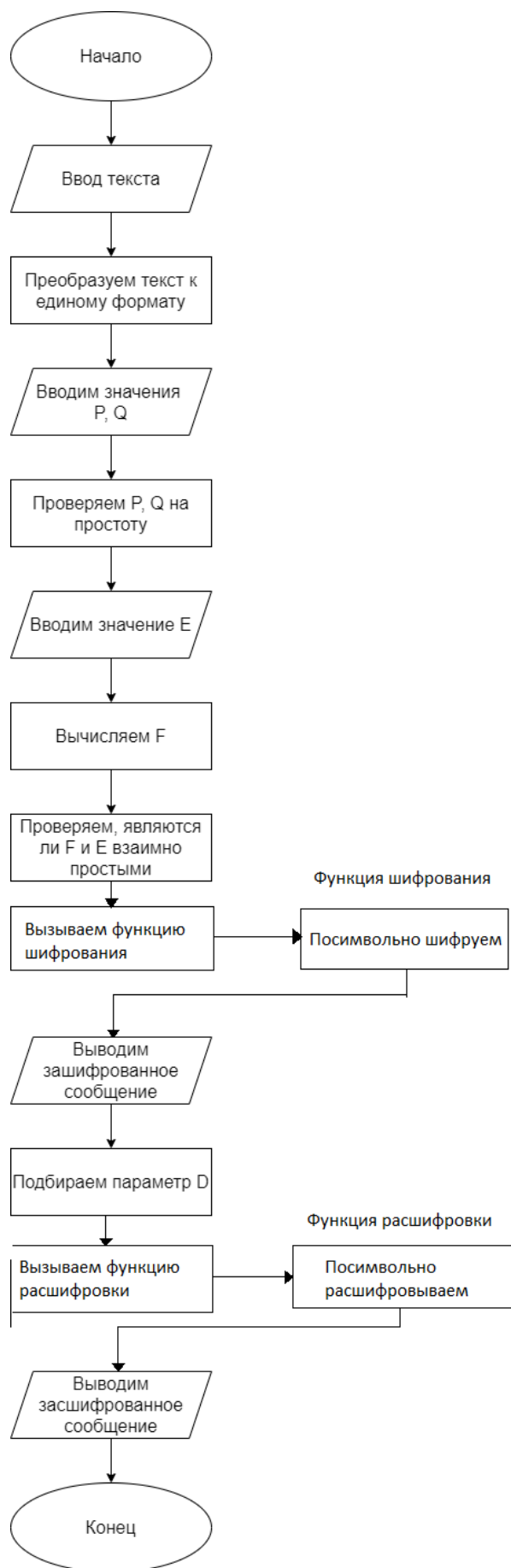
$$C_i = M_i^E \bmod N.$$

*Расшифрование:*

Получатель расшифровывает сообщение, возводя последовательно  $C_i$  в степень  $D$  по модулю  $N$ :

$$M_i = C_i^D \bmod N.$$

Блок-схема программы:





## Код программы с комментариями:

```
# ШИФР RSA
def prime_num(num): # проверка на простоту
    k = 0
    check = 1
    while check!=0:
        for i in range(2, num // 2+1):
            if (num % i == 0):
                k = k+1
        if (k <= 0):
            print("Число простое")
            check = 0
        elif(check!=0):
            print("Число не является простым")
            num = int(input('Введите значение параметра P(простое число): '))
            k = 0
    return num

def is_coprime(e, f): # проверка на взаимную простоту
    check = 1
    while check!=0:
        if(math.gcd(e, f) == 1):
            check = 0
            print('Числа ', e, 'и', f, ' взаимно простые')
        else:
            print('Числа ', e, 'и', f, ' НЕ взаимно простые')
            e = int(input('Введите новое значение параметра E: '))
    return (e)

def cryp_rsa(pr, n, e): #функция шифрования
    cryp_text = []
    for i in pr:
        cryp_text.append((alph.find(i) ** e) % n)
    return cryp_text

def dec_rsa(pr, n, e, f): #функция расшифровки
    n_pr = len(pr)
    decryp_text = ""
    for i in range(100000):
        if i * e % f == 1:
            d = i
            break
    for i in range(n_pr):
        # print((int(pr[i]) ** d) % n)
        decryp_text = decryp_text + alph[(int(pr[i]) ** d) % n]
    return decryp_text
# КОНЕЦ ШИФРА RSA
```

## Часть кода с вызовом функций:

```
elif code_cp == "13": # проверяем, какой шифр вызвал пользователь
    proverb = change_line(proverb) # вызываем функцию, которая преобразует строку

    p = int(input("Введите значение параметра P(простое число):"))
    p = prime_num(p) # проверяем параметр p на простоту
```

```

q = int(input("Введите значение параметра Q(простое число):"))
q = prime_num(q) # проверяем параметр q на простоту
n=p*q
f = (p-1)*(q-1)
print("Введите значение параметра E, взаимно простое", f, ": ")
e = int(input())
e = int(is_coprime(e, f)) # проверяем является ли e взаимно простое f

cryp_text = cryp_rsa(proverb, n, e) # вызываем функцию шифрования
print("Зашифровано(RSA)")
print(cryp_text)
decryp_text = dec_rsa(cryp_text, n, e, f) # вызываем функцию расшифровки
print("Расшифровано(RSA)")
print(decryp_text)

```

## Тестирование:

```

3.4 - 3.4.1
15 - Обмен ключами по алгоритму Diffie-Hellman
13
Введите значение параметра P(простое число):13
Число не является простым
Введите значение параметра P(простое число): 17
Число простое
Введите значение параметра Q(простое число):7
Число простое
Введите значение параметра E, взаимно простое 96 :
13
Зашифровано(RSA)
[109, 54, 58, 36, 0, 16, 4, 13, 54, 82, 58, 27, 54, 18, 43, 91, 82, 54, 13, 43, 18, 7, 17, 109, 56, 43, 21, 36, 73, 18, 54, 13, 18, 44, 45]
Расшифровано(RSA)
подписанное сообщение с открытым текстом

```



$$P = 2317 \quad \text{из этого следует:}$$

$$Q = 7 \quad n = P \cdot Q = 119$$

$$E = 13$$

Зашифруем несколько  
первых букв для проверки

Пескорок не может урешить  
своих котен.

$$\begin{array}{ll} A = 11 & A = 15 \\ E = 5 & A = 0 \\ O = 14 \end{array}$$

$$11^{13} \bmod 119 = 109$$

$$5^{13} \bmod 119 = 54$$

$$14^{13} \bmod 119 = 58$$

$$15^{13} \bmod 119 = 36$$

$$0^{13} \bmod 119 = 0$$

Работа с текстом не менее 1000 знаков:

© 2007 The Authors  
Journal compilation © 2007 Blackwell Publishing Ltd

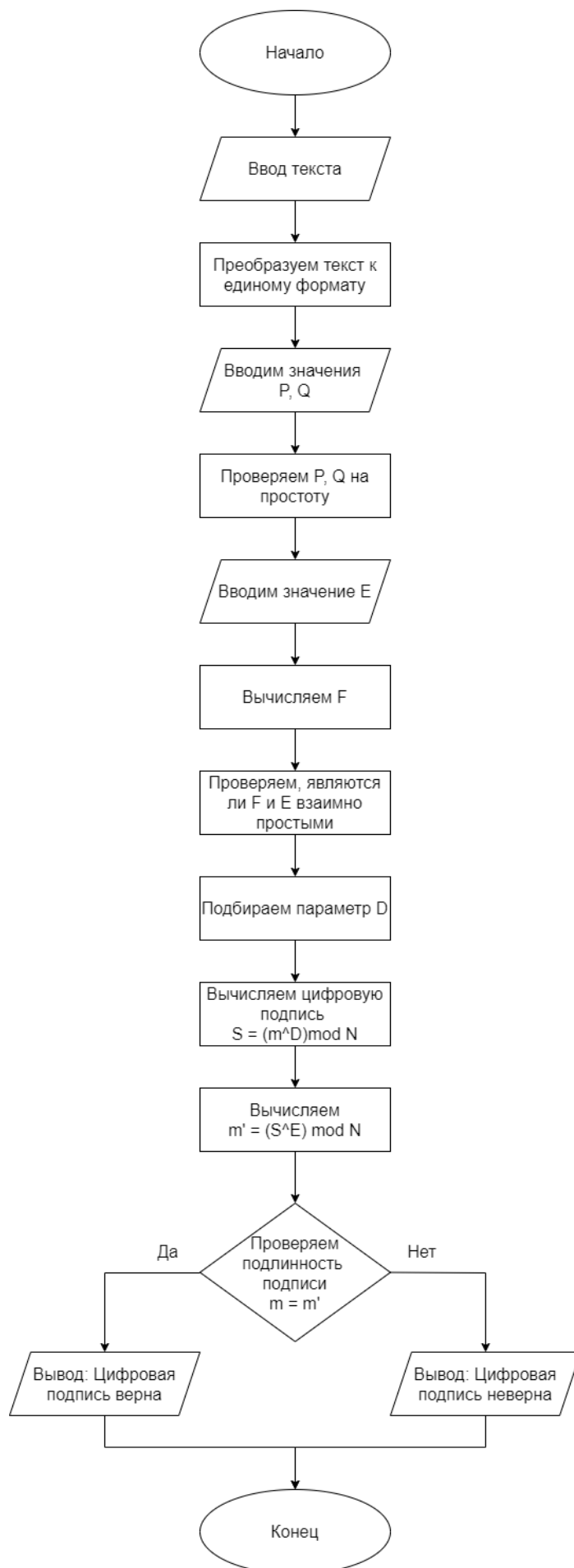
## Блок I: АЛГОРИТМЫ ЦИФРОВЫХ ПОДПИСЕЙ

### 24. Шифр RSA

Алгоритм шифра, описание:

1. Берутся два очень больших простых числа  $P$  и  $Q$  и находится произведение простых чисел  $N=P \times Q$  и функция Эйлера от этого произведения  $\phi(N)=(P-1) \times (Q-1)$ .
2. Выбирается случайное целое число  $E$ , взаимно простое с  $\phi(N)$ , и вычисляется
$$D=(1 \bmod \phi(N))/E.$$
3. Потом  $E$  и  $N$  публикуются как открытый ключ,  $D$  сохраняется в тайне.
4. Если  $M$  — сообщение, а  $h(M) = m$  — хеш-код сообщения, длина которого должна быть в интервале  $(1, N)$ , то электронная цифровая подпись  $S$  получается шифрованием хеш-кода сообщения  $m$  возведением в степень  $D$  по модулю  $N$ :  $S = m^D \bmod N$ .
5. Получателю отправляется сообщение  $M$  и подпись  $S$ .
6. Получатель сообщения хеширует  $M$ , получает хеш-код  $m'$ .
7. Проверяет подпись  $S$ : расшифровывает хеш-код, возведя  $S$  в степень  $E$  (число  $E$  ему известно) по модулю  $N$ :  $m = S^E \bmod N$ .
8. Сравнивает  $m$  и  $m'$ : если  $m = m'$  - подпись верна.

Блок-схема программы:



## Код программы с комментариями:

```
import random, math

# ПОСТОЯННЫЕ
alph = "абвгдежзийклмнопрстуфхцщщъыьэюя"

# ШИФР RSA
def prime_num(num): # проверка на простоту
    k = 0
    check = 1
    while check!=0:
        for i in range(2, num // 2+1):
            if (num % i == 0):
                k = k+1
        if (k <= 0):
            print("Число простое")
            check = 0
        elif(check!=0):
            print("Число не является простым")
            num = int(input('Введите значение параметра Р(простое число): '))
            k = 0
    return num

def is_coprime(e, f): # проверка на взаимную простоту
    check = 1
    while check!=0:
        if(math.gcd(e, f) == 1):
            check = 0
            print('Числа ', e, 'и', f, ' взаимно простые')
        else:
            print('Числа ', e, 'и', f, ' НЕ взаимно простые')
            e = int(input('Введите новое значение параметра E: '))
    return (e)

def hash_m(pr,p): # функция вычисления хэша
    h = h0 = 0
    for i in pr:
        h=((h0+alph.find(i))**2)%p
        h0=h
    return h

def cryp_prsa(pr, n, e, f): # функция вычисления цифровой подписи
    i = 0
    while i < n:
        if (i * e) % f == 1:
            d = i
            break
        i += 1

    print("Параметр d = ", d)
    s = (pr ** d) % n

    return s

def dec_prsa(s, e): # функция проверки цифровой подписи
    m = (s ** e) % n
    return m
```

```

# КОНЕЦ ШИФРА RSA

# proverb = "Леопард не может изменить своих пятен."
proverb = input("Введите текст для шифрования: ")

proverb = proverb.replace(" ", "")
proverb = proverb.lower()
while proverb.find(",") != -1:
    str1 = proverb[:proverb.find(",")]
    str2 = proverb[proverb.find(",")+1:]
    proverb = str1 + "зпт" + str2
while proverb.find(".") != -1:
    str1 = proverb[:proverb.find(".")]
    str2 = proverb[proverb.find(".")+1:]
    proverb = str1 + "тчк" + str2

p = int(input("Введите значение параметра P(простое число):"))
p = prime_num(p)
q = int(input("Введите значение параметра Q(простое число):"))
q = prime_num(q)
n=p*q
f = (p-1)*(q-1)
print("Введите значение параметра E, взаимно простое", f, ": ")
e = int(input())
e = int(is_coprime(e, f))


hash = hash_m(proverb, p) #вычисляем хэш

s = cryp_prsa(hash, n, e, f) #вычисляем цифровую подпись
print("Сообщение: ", proverb)
print("Электронная подпись: ", s)
m = dec_prsa(s, e) # подтверждаем цифровую подпись
if m == hash:
    print("Подтверждение электронной подписи")
    print("Хэш функции равны", m, "=", hash)
else:
    print("Подтверждение электронной подписи")
    print("Хэш функции не равны:", m, "!=" , hash)

```

Тестирование:

```
PS C:\Users\Полина\Desktop\5 семестр\Крипта> & C:/Users/П
Введите текст для шифрования: Леопард не может изменить с
Введите значение параметра P(простое число):17
Число простое
Введите значение параметра Q(простое число):13
Число простое
Введите значение параметра E, взаимно простое 192 :
19
Числа 19 и 192 взаимно простые
Параметр d = 91
Сообщение: леопарднеможетизменитьсясвоихпятентчк
Электронная подпись: 100
Подтверждение электронной подписи
Хэш функции равны 9 = 9
PS C:\Users\Полина\Desktop\5 семестр\Крипта> █
```





② Алгоритм шифра и дешифрации  
 RSA

Сообщение:  $P=17, Q=13, E=19, D=91$   
 $N=P \times Q = 221$

Леопард не может изменить  
 своих котен.

Рассчитаем эти сообщения!

ЛЕОПАРД	НЕ	МОЖЕТ	ИЗМЕНИ
11 5 14 15 0 16 4	13 5	12 14 6 5 18	8 7 12 5 13 8

ТЬ	СВОИХ	ПЯТЕН	ТЧК
18 28	17 2 14 8 21	15 31 18 5 13	18 23 10

$$121 \bmod 17 = 2$$

$$49 \bmod 17 = 15$$

$$\left(\frac{15+14}{29}\right)^2 \bmod 17 = 841 \bmod 17 = 8$$

$$\left(\frac{15+8}{23}\right)^2 \bmod 17 = 529 \bmod 17 = 2$$

$$4 \bmod 17 = 4$$

.....

$$(4+10)^2 \bmod 17 = 9$$

Ред эти орис сообщ. = 9

Работа с текстом не менее 1000 знаков:



```
P5 C:\Users\Polina\Desktop\5 семестр\Крипта & C:\Users\Polina\AppData\Local\Programs\Python\Python38-32\python.exe c:\Users\Polina\Desktop\podr.py
Введите текст для шифрования: Лидия Петровна. Бдительки. Мил, да был бдительки. У него были усы, шила и сердце. И он решил жениться. Он решил жениться, когда стукнет без пятнадцати девять. Ровно в восемь он сделал предложение графину с водой. Графин с водой согласился немедленно, но в пятнадцать минут девятого его унесли и выдали замуж за водопроводный кран. Дело было сделано, и графин вернулся на стол к бдительки уже замужней дамой. Было двадцать минут девятого. Времени оставалось мало, бдительки тогда сделал предложение очка. Очки были старые и неоднократно выодили замуж за уш. Очки подумали пять минут и согласились, но в этот момент их опять выдали замуж за уш. Было уже восемь часов двадцать пять минут. Тогда бдительки быстро сделал предложение юнга. Юнга тут же согласилась, и бдительки стал идти, когда же стукнет без пятнадцати девять. Сердце его очень громко колотилось. Тут его в затыл и накрыли подушкой, потому что детей уложили спать. И без пятнадцати девять бдительки неожиданно для себя женился на подушке.
Введите значение параметра P(простое число):23
Число простое
Введите значение параметра Q(простое число):17
Число простое
Введите значение параметра E, взаимно простое 352 :
13
Числа 13 и 352 взаимно простые
Параметр d = 325
Сообщение: лидияпетровна.бдительки.мил,дабылбдительки.унегобылиусы.шилаисердце.ионрешилжениться.онрешилжениться,когдастукнетбезпятнадцатидевять.ровновосемьонсделалпредложениеграфинусводой.графинсводойсогласилсянемедленно.новпятнацатьминутдевятогоегоунеслиивыдализамужзаводопроводныйкран.делобылосделано.играфинвернулсянастолкбдителькиужезамужнейдамой.былодвадцатьминутдевятого.времениоставалосмало.бдителькитогдасделалпредложениеочка.очкибылестарыеинесколькоразвыодилизамужзауш.очкиподумалипятьминутисогласились,новэтотмоментихопятьвыдализамужзауш.былоужеосемьчасовдвадцатьпятьминут.тогдабдителькибыстросделалпредложениеюнге.юнгатутжесогласилась.ибдителькисталидти,когдажестукнетбезпятнадцатидевять.сердцеегооченьгромкоколотилось.тутеговзатылокнакрылиподушкой.потомучтодетейуложилиспать.ибезпятнадцатидевятьбдителькинеожиданнодлясебяженилсянаподушке.
Электронная подпись: 58
Подтверждение электронной подписи
Хэш функции равны 6 = 6
```

## Блок J: СТАНДАРТЫ ЦИФРОВЫХ ПОДПИСЕЙ

### 28.ГОСТ Р 34.10-2012

Алгоритм шифра, описание:

#### 6.1 Формирование цифровой подписи

Для получения цифровой подписи под сообщением  $M \in V^*$  необходимо выполнить следующие действия (шаги) по алгоритму I:

Шаг 1 – вычислить хэш-код сообщения  $M : \bar{h} = h(M)$ . (14)

Шаг 2 – вычислить целое число  $\alpha$ , двоичным представлением которого является вектор  $\bar{h}$ , и определить

$$e \equiv \alpha \pmod{q}. \quad (15)$$

Если  $e = 0$ , то определить  $e = 1$ .

Шаг 3 – сгенерировать случайное (псевдослучайное) целое число  $k$ , удовлетворяющее неравенству

$$0 < k < q. \quad (16)$$

Шаг 4 – вычислить точку эллиптической кривой  $C = kP$  и определить

$$r \equiv x_c \pmod{q}, \quad (17)$$

где  $x_c$  –  $x$ -координата точки  $C$ .

Если  $r = 0$ , то вернуться к шагу 3.

Шаг 5 – вычислить значение

ГОСТ Р 34.10-2012

$$s \equiv (rd + ke) \pmod{q}. \quad (18)$$

Если  $s = 0$ , то вернуться к шагу 3.

Шаг 6 – вычислить двоичные векторы  $\bar{r}$  и  $\bar{s}$ , соответствующие  $r$  и  $s$ , и определить цифровую подпись  $\zeta = (\bar{r} \parallel \bar{s})$  как конкатенацию двух двоичных векторов.

Исходными данными этого процесса являются ключ подписи  $d$  и подписываемое сообщение  $M$ , а выходным результатом – цифровая подпись  $\zeta$ .

Схема процесса формирования цифровой подписи приведена на рисунке 2.

Блок-схема программы:



## Код программы с комментариями:

```
from os import urandom

from codecs import getdecoder
from codecs import getencoder
from sys import version_info

from pygost.utils import hexdec, hexenc, long2bytes, bytes2long
from pygost import gost34112012512

def modinvert(a, n):
    """ Modular multiplicative inverse
    :returns: inverse number. -1 if it does not exist
    Realization is taken from:
    https://en.wikipedia.org/wiki/Extended\_Euclidean\_algorithm
    """
    if a < 0:
        #  $k^{-1} = p - (-k)^{-1} \bmod p$ 
        return n - modinvert(-a, n)
    t, newt = 0, 1
    r, newr = n, a
    while newr != 0:
        quotient = r // newr
        t, newt = newt, t - quotient * newt
        r, newr = newr, r - quotient * newr
    if r > 1:
        return -1
    if t < 0:
        t = t + n
    return t

MODE2SIZE = {
    2001: 32,
    2012: 64,
}

class GOST3410Curve(object):
    """ GOST 34.10 validated curve
    >>> curve = CURVES["id-GostR3410-2001-TestParamSet"]
    >>> prv = prv_unmarshal(urandom(32))
    >>> signature = sign(curve, prv, GOST341194(data).digest())
    >>> pub = public_key(curve, prv)
    >>> verify(curve, pub, GOST341194(data).digest(), signature)
    True
    :param long p: characteristic of the underlying prime field
    :param long q: elliptic curve subgroup order
    :param long a, b: coefficients of the equation of the elliptic curve in
        the canonical form
    :param long x, y: the coordinate of the point P (generator of the
        subgroup of order q) of the elliptic curve in
        the canonical form
    :param long e, d: coefficients of the equation of the elliptic curve in
```

```

        the twisted Edwards form
"""
def __init__(self, p, q, a, b, x, y, e=None, d=None):
    self.p = p
    self.q = q
    self.a = a
    self.b = b
    self.x = x
    self.y = y
    self.e = e
    self.d = d
    r1 = self.y * self.y % self.p
    r2 = ((self.x * self.x + self.a) * self.x + self.b) % self.p
    if r1 != self.pos(r2):
        raise ValueError("Invalid parameters")
    self._st = None

def pos(self, v):
    """Make positive number
    """
    if v < 0:
        return v + self.p
    return v

def _add(self, p1x, p1y, p2x, p2y): #находим точки на эллиптической кривой
    if p1x == p2x and p1y == p2y:
        t = ((3 * p1x * p1x + self.a) * modinvert(2 * p1y, self.p)) % self.p
    else:
        tx = self.pos(p2x - p1x) % self.p
        ty = self.pos(p2y - p1y) % self.p
        t = (ty * modinvert(tx, self.p)) % self.p
    tx = self.pos(t * t - p1x - p2x) % self.p
    ty = self.pos(t * (p1x - tx) - p1y) % self.p
    return tx, ty

def exp(self, degree, x=None, y=None): #находим точки на эллиптической кривой
    x = x or self.x
    y = y or self.y
    tx = x
    ty = y
    if degree == 0:
        raise ValueError("Bad degree value")
    degree -= 1
    while degree != 0:
        if degree & 1 == 1:
            tx, ty = self._add(tx, ty, x, y)
            degree = degree >> 1
        x, y = self._add(x, y, x, y)
    return tx, ty

def st(self):
    """Compute s/t parameters for twisted Edwards curve points conversion
    """
    if self.e == None or self.d == None:
        raise ValueError("non twisted Edwards curve")
    if self._st != None:
        return self._st
    self._st = (
        self.pos(self.e - self.d) * modinvert(4, self.p) % self.p,
        (self.e + self.d) * modinvert(6, self.p) % self.p,
    )

```

[illegible]

```
CURVES["id-tc26-gost-3410-2012-256-paramSetD"] = CURVES["id-GostR3410-2001-CryptoPro-C-ParamSet"]
```

```
DEFAULT_CURVE = CURVES["id-GostR3410-2001-CryptoPro-A-ParamSet"]
```

```
def public_key(curve, prv): # Формируем публичный ключ, исходя из секретного
    """ Generate public key from the private one
    :param GOST3410Curve curve: curve to use
    :param long prv: private key
    :returns: public key's parts, X and Y
    :rtype: (long, long)
    """
    return curve.exp(prv)
```

```
def sign(curve, prv, digest, rand=None, mode=2001): # Вычисление цифровой подписи
    size = MODE2SIZE[mode]
    q = curve.q
    e = bytes2long(digest) % q
    if e == 0:
        e = 1
    while True:
        if rand is None:
            rand = urandom(size)
        elif len(rand) != size:
            raise ValueError("rand length != %d" % size) #обработка ошибки
        k = bytes2long(rand) % q
        if k == 0:
            continue
        r, _ = curve.exp(k) # вычисляем координату, вторая координата не нужна
        r %= q
        if r == 0:
            continue
        d = prv * r
        k *= e
        s = (d + k) % q
        if s == 0:
            continue
        break
    return long2bytes(s, size) + long2bytes(r, size)
```

```
def verify(curve, pub, digest, signature, mode=2012): #проверка подлинности подписи
    """ Verify provided digest with the signature
    :param GOST3410Curve curve: curve to use
    :type pub: (long, long)
    :param digest: digest needed to check
    :type digest: bytes, 32 or 64 bytes
    :param signature: signature to verify with
    :type signature: bytes, 64 or 128 bytes
    :rtype: bool
    """
    size = MODE2SIZE[mode]
    if len(signature) != size * 2:
        print("Invalid signature length")
    q = curve.q
    p = curve.p
    s = bytes2long(signature[:size])
    r = bytes2long(signature[size:])
    if r <= 0 or r >= q or s <= 0 or s >= q:
```

```

        return False
    e = bytes2long(digest) % q
    if e == 0:
        e = 1
    v = modinvert(e, q)
    z1 = s * v % q
    z2 = q - r * v % q
    p1x, p1y = curve.exp(z1)
    q1x, q1y = curve.exp(z2, pub[0], pub[1])
    lm = q1x - p1x
    if lm < 0:
        lm += p
    lm = modinvert(lm, p)
    z1 = q1y - p1y
    lm = lm * z1 % p
    lm = lm * lm % p
    lm = lm - p1x - q1x
    lm = lm % p
    if lm < 0:
        lm += p
    lm %= q
    return lm == r

def prv_unmarshal(prv): # Преобразуем ключ в строку байт
    """Unmarshal private key
    :param bytes prv: serialized private key
    :rtype: long
    """
    return bytes2long(prv[::-1])

def pub_marshal(pub, mode=2012): # Для корректной работы с байтами ключа
    size = MODE2SIZE[mode]
    return (long2bytes(pub[1], size) + long2bytes(pub[0], size))

def pub_unmarshal(pub, mode=2012): # Для корректной работы с байтами ключа
    size = MODE2SIZE[mode]
    pub = pub[::-1]
    return (bytes2long(pub[size:]), bytes2long(pub[:size]))

curve = CURVES["id-tc26-gost-3410-12-512-paramSetA"]

pr = input("Введите текст для шифрования:")
data = pr.encode('utf-8')
key = urandom(64)

prv_raw = key
prv = prv_unmarshal(prv_raw)
pub = public_key(curve, prv)
# print(pub)
print("Открытый ключ:", hexenc(pub_marshal(pub)))
dgst = gost34112012512.new(data).digest() #хэширование данных, к сожалению этой функции
нет в открытом доступе.
signature = sign(curve, prv, dgst)
print("Подпись:", hexenc(signature))

```



```

b = verify(curve, pub, dgst, signature)
if b is True:
    print('Подпись прошла проверку')
else:
    print('Подпись не прошла проверку')

```

## Тестирование:

Для тестирования использовался упрощенный код с вызовом функций из готовой библиотеки. Используемый код из библиотеки есть выше.

## Код для тестирования по ГОСТ:

```

from pygost.gost3410 import GOST3410Curve
from pygost.gost3410 import sign
from pygost.utils import bytes2long
from pygost.utils import hexdec
from pygost.utils import hexenc

curve = GOST3410Curve(
    p=3623986102229003635907788753683874306021320925534678605086546150450856166624002482588
482022271496854025090823603058735163734263822371964987228582907372403,
    q=3623986102229003635907788753683874306021320925534678605086546150450856166623969164898
305032863068499961404079437936585455865192212970734808812618120619743,
    a=7,
    b=1518655069210828534508950034714043154928747527740206436194018823352809982443793732829
756914785974674866041605397883677596626326413990136959047435811826396,
    x=1928356944067022849399309401243137598997786635459507974357075491307766592685835441065
557681003184874819658004903212332884252335830250729527632383493573274,
    y=2288728693371972859970012155529478416353562327329506180314497425931102860301572814141
997072271708807066593850650334152381857347798885864807605098724013854,
)
prv =
bytes2long(hexdec("0BA6048AADAE241BA40936D47756D7C93091A0E8514669700EE7508E508B102072E8123B
2200A0563322DAD2827E2714A2636B7BFD18AADFC62967821FA18DD4"))
digest =
hexdec("3754F3CFACC9E0615C4F4A7C4D8DAB531B09B6F9C170C533A71D147035B0C5917184EE536593F441433
9976C647C5D5A407ADEDB1D560C4FC6777D2972075B8C")
rand =
hexdec("0359E7F4B1410FEACC570456C6801496946312120B39D019D455986E364F365886748ED7A44B3E79443
4006011842286212273A6D14CF70EA3AF71BB1AE679F1")
signature = sign(curve, prv, digest, rand)
r =
"2f86fa60a081091a23dd795e1e3c689ee512a3c82ee0dcc2643c78eea8fcacd35492558486b20f1c9ec197c906
99850260c93bcbcd9c5c3317e19344e173ae36"
s =
"1081b394696ffe8e6585e7a9362d26b6325f56778aadbc081c0bfbe933d52ff5823ce288e8c4f362526080df7f
70ce406a6eeb1f56919cb92a9853bde73e5b4a"

print("p:", curve.p)
print("q:", curve.q)
print("a:", curve.a)
print("b:", curve.b)
print("x:", curve.x)

```

```

print("y:", curve.y)
print("d:", prv)
print("e:", hexenc(digest))
print("k:", hexenc(rand))
print("r:", r)
print("s:", s)
print("Вычисленная подпись:", hexenc(signature))
print("Подпись из примера ГОСТ: ", s + r)
print()

```

```

p: 36239861822298836159877887536838743888213289255346786958865461584588561666248824825884828222714968548258888236838587351637342638223719648672285829873
72483
q: 36239861822298836159877887536838743888213289255346786958865461584588561666239991648883858128638684999914848794379365854358851922129787348888126181286
18743
a: 7
b: 151885888718828534588588834714843154828747527748286436194818823532888834437937328297569147858746748668416853978836775866263264139981389588474358118
26396
x: 192835884887822849399388481343137588877866345958797435787548138776659268583544186555788188184874819658884883213328842523368382587295276323834835
73274
y: 2288728893371972859978812155529478416353623273295861883144974258311828888815728141419978722717888878868838888834152381857347798885884887888887248
13854
d: 61888188413637888219538153238847583888845518888515628823881353548888638178225538368839342337379857665527955116827387825884883744876612118846687
5888
e: 3754f3cfacc9e8e15c4f4a7c488ab531b80b6f9c178c533a71d14793588c5917184e538893f4414399976c847c5e5a487adedb1d568c4fc6777d2972875b8c
k: 8359e7f4b5418faacc578455cc8884888488112128b39d819d455888e384f365888748ed7a44b3e794434888811842188212273a8d14cf78ea3af71bb1a8579f1
r: 2f88fad8a881891a23dd795e1e3c888ee512a3c82ee8d8c2643c78ee8f8acd35492558488828f1c9ec197c98898858888c93bcbcd8c5c3317e19344e173ae36
s: 1881b394696ff8e8e585e7a93d2d26b325f56778aadbc881c8bf8e933d52ff5823ce288e8c4f362526888b77f78ca48daeeb1f58819cb92a8853bde73e5b4a
Подпись из примера ГОСТ: 1881b394696ff8e8e585e7a93d2d26b325f56778aadbc881c8bf8e933d52ff5823ce288e8c4f362526888b77f78ca48daeeb1f58819cb92a8853bde73e5b4
a2f88f8a881891a23dd795e1e3c888ee512a3c82ee8d8c2643c78ee8f8acd35492558488828f1c9ec197c98898858888c93bcbcd8c5c3317e19344e173ae36
Вычисленная подпись: 1881b394696ff8e8e585e7a93d2d26b325f56778aadbc881c8bf8e933d52ff5823ce288e8c4f362526888b77f78ca48daeeb1f58819cb92a8853bde73e5b4a2f
88f8a881891a23dd795e1e3c888ee512a3c82ee8d8c2643c78ee8f8acd35492558488828f1c9ec197c98898858888c93bcbcd8c5c3317e19344e173ae36

```

## Тестирование на текстах:

Введите текст для шифрования: Леопард не может изменить своих пятен.  
 Открытый ключ: abdfe82f86d1dfd5e2ca1067c88b417cbe16ee8b11568ef0b448141e40f74b28574367987801c8ffbf819d11a54038db44f2e94e8d98cfff809c1508b8c4f64b457167d10187ec5f07f9fb0b8db9af4ae4f3ca1a1059a492dcf15a5a2ef71535682e7b088e14caf162a715f58e1e7f5f044b28c7b026e888863a0d4e15a2c9  
 Подпись: 3cfb14b3f06804a4ffe447053c7ce87be07d88b8e96cc9e8aa1e168ed4bbcf95c89ef3532a700632dded54773cb54711e00192ffab9218abb03f88bc085bd8683158c98f040b8b38801e853df999c7efe53837f88eccc1a89dc8ca4a6acbc3b68253105f71a745af0feeb148fe9cbdf40874252a21a727b73868641bfc4f02e7  
 Подпись прошла проверку

Введите текст для шифрования: Будильник Петрушевская, Будильник. Жил, да был будильник. У него были усы, шляпа и сердце. И он решил жениться. Он решил жени  
 тыся, когда стукнет без пятнадцати девять. Ровно в восемь он сделал предложение графине с водой. Графин с водой согласился немедленно, но в пятнадцать н  
 минут девятиго его унесли и выдали замуж за водопроводный кран. Дело было сделано, и графин вернулся на стол к будильнику уже замужней дамой. Было двадца  
 ть минут девятюго. Времени оставалось мало. Будильник тогда сделал предложение очкам. Очки были старые и неоднократно выходили замуж за уш. Очки подум  
 ли пять минут и согласились, но в этот момент их опять выдали замуж за уш. Было уже восемь часов двадцать пять минут. Тогда будильник быстро сделал пре  
 дложение юнге. Книга тут же согласилась, и будильник стал кдаты, когда же стукнет без пятнадцати девять. Сердце его очень громко колотилось. Тут его вз  
 али и накрыли подушкой, потому что детей уложили спать. И без пятнадцати девять будильник неожиданно для себя женился на подушке.  
 Открытый ключ: ab200fb37fc84057448881867a5939a543c2a2ba04fd979dcd08028237f443609d8cc8dc2e8b56885966159135eb1781c9864e7ea3db80429d3861b69ea383621977c56  
 c3b468ffe718843317bd572388631a06e8ff88d54c18328b8a89f3dea18463a2434fb394ca87d0d6e128aa4cee18adbed1abb0405841ad78867bf74  
 Подпись: 1653481fa89747b2111687503b8b889da418388118234d28eabc8fee1e8b9a516ea9c1db7a61eaadff0f38d233f7fec1f118c261422d98e9dc45b77a767253a7f9e89a2d3433f25888  
 e5ed1498b9bf43ced40d2d358b499ddfa9498c8964aadca6daf9cc437588916e25802fa92192868bff53032a06d7929a8deca7e54688a8233  
 Подпись прошла проверку

## БЛОК К: ОБМЕН КЛЮЧАМИ

### 29. Обмен ключами по Diffi-Hellman

Алгоритм шифра, описание:

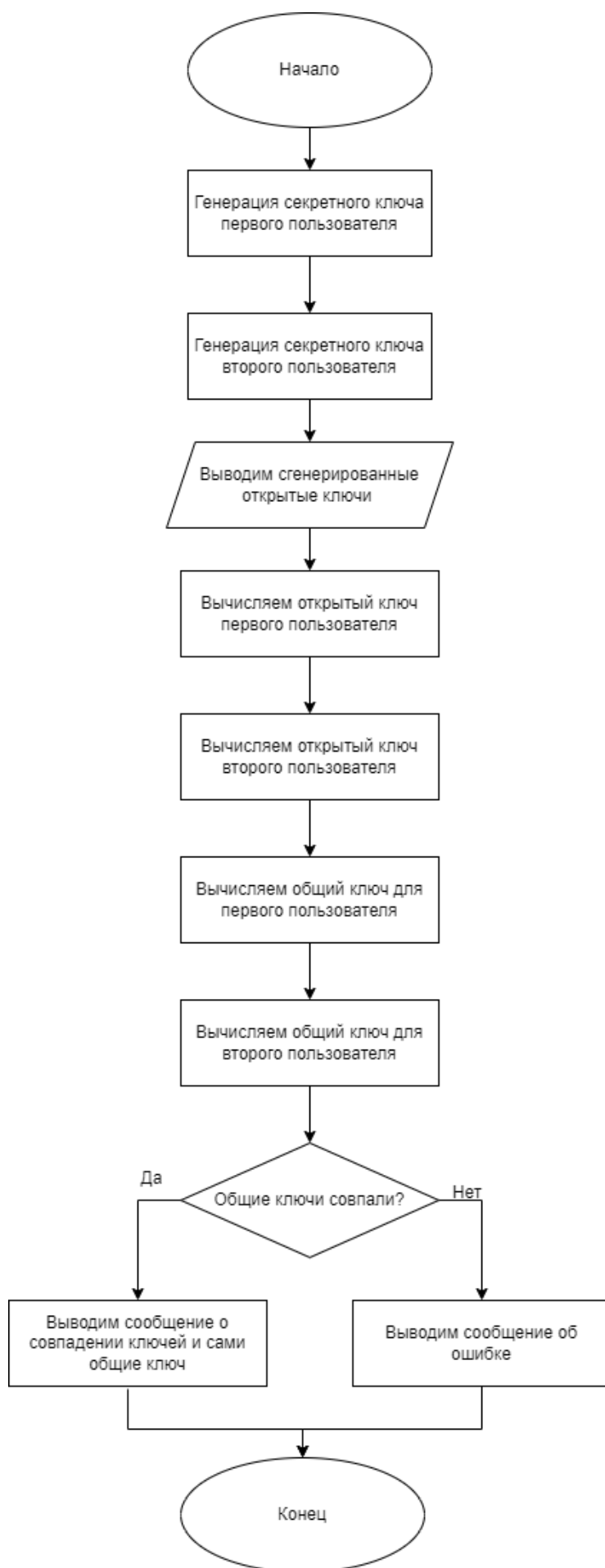
1. Определить секретные ключи пользователей  $K_A$  и  $K_B$ .
2. Для этого каждый пользователь независимо выбирает случайные числа из интервала  $(2, \dots, n-1)$ .
3. Вычислить открытые ключи пользователей  $Y_A$  и  $Y_B$ :  
$$Y = a^K \bmod n$$
4. Обменяться ключами  $Y_A$  и  $Y_B$  по открытому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

Доказательство:

$$A: \quad Y_B^{K_A}(\bmod n) = (a^{K_B})^{K_A} \bmod n = a^{K_A \cdot K_B} \bmod n = K.$$

$$B: \quad Y_A^{K_B}(\bmod n) = (a^{K_A})^{K_B} \bmod n = a^{K_B \cdot K_A} \bmod n = K.$$

Блок-схема программы:



## Код программы с комментариями:

```
# ОБМЕН КЛЮЧАМИ ПО АЛГОРИТМУ DIFFIE-HELLMAN
def exchange_key(n, a):
    key_a = random.randint(2, n-1) # генерируем Секретный ключ первого пользователя
    key_b = random.randint(2, n-1) # генерируем Секретный ключ второго пользователя
    print("Секретный ключ первого пользователя", key_a)
    print("Секретный ключ второго пользователя", key_b)
    y_a = (a ** key_a) % n # вычисляем открытый ключ 1-ого пользователя
    y_b = (a ** key_b) % n # вычисляем открытый ключ 2-ого пользователя
    key_ab = (y_b ** key_a) % n # вычисляем общий ключ для 1-ого пользователя
    key_ba = (y_a ** key_b) % n # вычисляем общий ключ для 2-ого пользователя
    if key_ab == key_ba:
        print("Обмен ключами работает, общие секретные ключи совпадают")
        print(key_ab, "=", key_ba)
    else:
        print("Какая-то ошибка;(")
# КОНЕЦ ОБМЕНА КЛЮЧАМИ DIFFIE-HELLMAN
```

## Часть кода с вызовом функций:

```
elif code_cp == "15": # проверяем, какой шифр вызвал пользователь
    print("Введите а и n , удовлетворяющие условию  $1 < a < n$  \n Введите а")
    a = int(input())
    print("Введите n")
    n = int(input())

    if (a > 1) and (a < n):
        exchange_key(n, a)
    else:
        print("Некорректно введены данные")
```

## Тестирование:

ШИФР ПЕРСТАНОВКИ

9 - Шифр Вертикальной Перестановки

ШИФР ГАММИРОВАНИЯ

10- Одноразовый блокнот Шеннона

ПОТОЧНЫЙ ШИФР

11 - A5/1

КОМБИНАЦИОННЫЙ ШИФР

12 - Кузнечик

АССИМЕТРИЧНЫЕ ШИФРЫ(ГЕНЕРАЦИЯ ЦИФРОВОЙ ПОДПИСИ)

13 - RSA

14 - Elgamal

15 - Обмен ключами по алгоритму Diffie-Hellman

15

Введите а и n , удовлетворяющие условию  $1 < a < n$

Введите а

35

Введите n

48

Секретный ключ первого пользователя 13

Секретный ключ второго пользователя 38

Обмен ключами работает, общие секретные ключи совпадают

25 = 25

PS C:\Users\Polina\Desktop\E\_состав\Крипто\ ■

○ Обмен ключами по алгоритму Diffie - Hellman

$$K_A = 13 \quad R = 35$$

$$K_B = 38 \quad n = 48$$

$$Y_A = 35^{13} \bmod 48 = 35$$

$$Y_B = 35^{38} \bmod 48 = 25$$

$$K_{AB} = 25^{13} \bmod 48 = 25$$

$$K_{BA} = 35^{38} \bmod 48 = 25$$

$$K_{AB} = K_{BA} = K = 25$$