Приложение 2. Демонстрация работы программы

Часть 1. Шифр простой замены

1.1 Программное зашифрование / расшифрование

In [1]:
```
# Програма шифра простой замены реализована
# в модуле simple_cipher классе SimpleCipher

!ls -l
```

```
total 680
-rw-r--r-- 1 evgeny evgeny 140344 May 19 19:02 decrypt.ipynb
-rw-r--r-- 1 evgeny evgeny  30772 May 27 18:47 demo.ipynb
-rw-r--r-- 1 evgeny evgeny  26671 May 19 16:44 english.jpg
drwxr-xr-x 2 evgeny evgeny   4096 May 19 17:53 __pycache__
-rw-r--r-- 1 evgeny evgeny   7998 May 19 17:53 simple_cipher.py
drwxr-xr-x 4 evgeny evgeny   4096 May 17 14:52 venv
-rw-r--r-- 1 evgeny evgeny 471411 May 28 17:53 отчет.docx
```

In [2]:
```
# Импортирую класс для демонстрации его работы

from simple_cipher import SimpleCipher
```

In [3]:
```
# Объявлю объект класса и выведу на экран его атрибуты

encryptor = SimpleCipher()

# Алфавит шифрования
encryptor.alphabet
```

Out[3]:
```
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~ '
```

In [4]:
```
# Модуль алгоритма шифрования
encryptor.module
```

Out[4]:
```
95
```

In [5]:
```
# Так как классу не был передан ключ шифрования (Таблица замены)
# то объект создался с ключом по-умолчанию
print(encryptor.secret_key)
```

```
[[29, 44, 65, 31, 48, 4, 92, 84, 10, 37, 90, 9, 66, 93, 25, 17, 47, 35, 33, 20, 23, 19, 63, 30, 24, 75, 6
8, 27, 1, 62, 59, 76, 7, 50, 16, 0, 5, 26, 22, 8, 85, 80, 54, 74, 89, 39, 36, 57, 55, 18, 40, 69, 43, 28,
86, 71, 91, 6, 78, 87, 12, 41, 15, 34, 56, 70, 58, 52, 21, 2, 60, 82, 67, 45, 49, 53, 11, 88, 73, 13, 79,
42, 46, 64, 61, 38, 94, 77, 83, 72, 14, 81, 51, 32, 3], [35, 28, 69, 94, 5, 36, 57, 32, 39, 11, 8, 76, 60,
79, 90, 62, 34, 15, 49, 21, 19, 68, 38, 20, 24, 14, 37, 27, 53, 0, 23, 3, 93, 18, 63, 17, 46, 9, 85, 45, 5
0, 61, 81, 52, 1, 73, 82, 16, 4, 74, 33, 92, 67, 75, 42, 48, 64, 47, 66, 30, 70, 84, 29, 22, 83, 2, 12, 7
2, 26, 51, 65, 55, 89, 78, 43, 25, 31, 87, 58, 80, 41, 91, 71, 88, 7, 40, 54, 59, 77, 44, 10, 56, 6, 13, 8
6]]
```

In [6]:
```
# Создам новый ключ (таблицу замены) и передам
# его объекту класса для шифрования

# import random

# Вот например таблица случайных размещений чисел от 0 до модуля
# random_nums = random.sample(range(encryptor.module), encryptor.module)
# print(random_nums)

random_nums = [
    58, 94, 86, 65, 92, 51, 25, 66, 33, 2, 49, 53, 61, 47, 59, 91, 26, 6,
    87, 36, 39, 55, 81, 84, 88, 56, 80, 7, 23, 52, 83, 21, 73, 76, 82, 77,
    78, 41, 67, 37, 93, 29, 13, 30, 85, 20, 3, 48, 15, 54, 69, 22, 17, 62,
    46, 75, 45, 28, 16, 9, 14, 32, 34, 89, 50, 27, 40, 63, 68, 90, 19, 74,
    44, 24, 72, 35, 1, 11, 64, 0, 38, 42, 5, 43, 79, 31, 8, 60, 71, 18, 57,
    10, 70, 12, 4]

print(random_nums)
```

```
[58, 94, 86, 65, 92, 51, 25, 66, 33, 2, 49, 53, 61, 47, 59, 91, 26, 6, 87, 36, 39, 55, 81, 84, 88, 56, 80,
7, 23, 52, 83, 21, 73, 76, 82, 77, 78, 41, 67, 37, 93, 29, 13, 30, 85, 20, 3, 48, 15, 54, 69, 22, 17, 62,
46, 75, 45, 28, 16, 9, 14, 32, 34, 89, 50, 27, 40, 63, 68, 90, 19, 74, 44, 24, 72, 35, 1, 11, 64, 0, 38, 4
2, 5, 43, 79, 31, 8, 60, 71, 18, 57, 10, 70, 12, 4]
```

In [7]:
```
# Теперь создан обратный массив random_nums_inv для
# более удобного обратного преобразования

# random_nums_inv = [0 for _ in range(encryptor.module)]

# for num, i in enumerate(random_nums):
#     random_nums_inv[i] = num

random_nums_inv = [
    79, 76, 9, 46, 94, 82, 17, 27, 86, 59, 91, 77, 93, 42, 60, 48, 58, 52,
    89, 70, 45, 31, 51, 28, 73, 6, 16, 65, 57, 41, 43, 85, 61, 8, 62, 75,
    19, 39, 80, 20, 66, 37, 81, 83, 72, 56, 54, 13, 47, 10, 64, 5, 29, 11,
    49, 21, 25, 90, 0, 14, 87, 12, 53, 67, 78, 3, 7, 38, 68, 50, 92, 88,
    74, 32, 71, 55, 33, 35, 36, 84, 26, 22, 34, 30, 23, 44, 2, 18, 24, 63,
    69, 15, 4, 40, 1]

print(random_nums_inv)
```

```
[79, 76, 9, 46, 94, 82, 17, 27, 86, 59, 91, 77, 93, 42, 60, 48, 58, 52, 89, 70, 45, 31, 51, 28, 73, 6, 16,
65, 57, 41, 43, 85, 61, 8, 62, 75, 19, 39, 80, 20, 66, 37, 81, 83, 72, 56, 54, 13, 47, 10, 64, 5, 29, 11,
49, 21, 25, 90, 0, 14, 87, 12, 53, 67, 78, 3, 7, 38, 68, 50, 92, 88, 74, 32, 71, 55, 33, 35, 36, 84, 26, 2
2, 34, 30, 23, 44, 2, 18, 24, 63, 69, 15, 4, 40, 1]
```

In [8]:
```
# Теперь оба массива можно передать классу SimpleCipher

encryptor = SimpleCipher([random_nums, random_nums_inv])
```

In [9]:
```
# Проверим что класс принял новый ключ для шифрования данных

print(encryptor.secret_key)
```

```
[[58, 94, 86, 65, 92, 51, 25, 66, 33, 2, 49, 53, 61, 47, 59, 91, 26, 6, 87, 36, 39, 55, 81, 84, 88, 56, 8
0, 7, 23, 52, 83, 21, 73, 76, 82, 77, 78, 41, 67, 37, 93, 29, 13, 30, 85, 20, 3, 48, 15, 54, 69, 22, 17, 6
2, 46, 75, 45, 28, 16, 9, 14, 32, 34, 89, 50, 27, 40, 63, 68, 90, 19, 74, 44, 24, 72, 35, 1, 11, 64, 0, 3
8, 42, 5, 43, 79, 31, 8, 60, 71, 18, 57, 10, 70, 12, 4], [79, 76, 9, 46, 94, 82, 17, 27, 86, 59, 91, 77, 9
3, 42, 60, 48, 58, 52, 89, 70, 45, 31, 51, 28, 73, 6, 16, 65, 57, 41, 43, 85, 61, 8, 62, 75, 19, 39, 80, 2
0, 66, 37, 81, 83, 72, 56, 54, 13, 47, 10, 64, 5, 29, 11, 49, 21, 25, 90, 0, 14, 87, 12, 53, 67, 78, 3, 7,
38, 68, 50, 92, 88, 74, 32, 71, 55, 33, 35, 36, 84, 26, 22, 34, 30, 23, 44, 2, 18, 24, 63, 69, 15, 4, 40,
1]]
```

In [10]:
```
# Кажется что всё правильно!
# Приступим к зашифрованию

# Определяем текст для шифрования

open_text = "I have a Dream!"

# Шифруем
encrypted_text = encryptor.encrypt(open_text)
encrypted_text
```

Out[10]:
```
'?e%63}e6e0g}69I'
```

In [11]:
```
# Расшифруем
decrypted_text = encryptor.decrypt(encrypted_text)
print(decrypted_text)
bool(decrypted_text == open_text)
```

```
I have a Dream!
```

Out[11]:
```
True
```

1.2 Зашифрование / расшифрование вручную

Итак, имеем:

- Открытый текст ***"I have a Dream!"*** который требуется зашифровать.
- Алфиавит

  ```
  abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~
  ```

- Ключ шифрования в виде матрицы замены символов

  ```
  [[58, 94, 86, 65, 92, 51, 25, 66, 33, 2, 49, 53, 61, 47, 59, 91, 26, 6, 87, 36, 39, 55, 81, 84, 88,
  56, 80, 7, 23, 52, 83, 21, 73, 76, 82, 77, 78, 41, 67, 37, 93, 29, 13, 30, 85, 20, 3, 48, 15, 54, 6
  9, 22, 17, 62, 46, 75, 45, 28, 16, 9, 14, 32, 34, 89, 50, 27, 40, 63, 68, 90, 19, 74, 44, 24, 72, 3
  5, 1, 11, 64, 0, 38, 42, 5, 43, 79, 31, 8, 60, 71, 18, 57, 10, 70, 12, 4],
  [79, 76, 9, 46, 94, 82, 17, 27, 86, 59, 91, 77, 93, 42, 60, 48, 58, 52, 89, 70, 45, 31, 51, 28, 73,
  6, 16, 65, 57, 41, 43, 85, 61, 8, 62, 75, 19, 39, 80, 20, 66, 37, 81, 83, 72, 56, 54, 13, 47, 10, 6
  4, 5, 29, 11, 49, 21, 25, 90, 0, 14, 87, 12, 53, 67, 78, 3, 7, 38, 68, 50, 92, 88, 74, 32, 71, 55,
  33, 35, 36, 84, 26, 22, 34, 30, 23, 44, 2, 18, 24, 63, 69, 15, 4, 40, 1]]
  ```

- Для удобства также буду пользоваться таблицей индексов символов алфавита

```
{'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4, 'f': 5, 'g': 6, 'h': 7, 'i': 8, 'j': 9, 'k': 10, 'l': 11,
'm': 12, 'n': 13, 'o': 14, 'p': 15, 'q': 16, 'r': 17, 's': 18, 't': 19, 'u': 20, 'v': 21, 'w': 22,
'x': 23, 'y': 24, 'z': 25, 'A': 26, 'B': 27, 'C': 28, 'D': 29, 'E': 30, 'F': 31, 'G': 32, 'H': 33,
'I': 34, 'J': 35, 'K': 36, 'L': 37, 'M': 38, 'N': 39, 'O': 40, 'P': 41, 'Q': 42, 'R': 43, 'S': 44,
'T': 45, 'U': 46, 'V': 47, 'W': 48, 'X': 49, 'Y': 50, 'Z': 51, '0': 52, '1': 53, '2': 54, '3': 55,
'4': 56, '5': 57, '6': 58, '7': 59, '8': 60, '9': 61, '!': 62, '"': 63, '#': 64, '$': 65, '%': 66,
'&': 67, "'": 68, '(': 69, ')': 70, '*': 71, '+': 72, ',': 73, '-': 74, '.': 75, '/': 76, ':': 77,
';': 78, '<': 79, '=': 80, '>': 81, '?': 82, '@': 83, '[': 84, '\\': 85, ']': 86, '^': 87, '_': 88,
'`': 89, '{': 90, '|': 91, '}': 92, '~': 93, ' ': 94}
```

Для зашифрования требуется заменить каждый исмвол из открытого текста символом из того же алфавита в соответствии с нулевым вектором матрицы замены символов. Например, если первый символ открытого текста I имеет индекс: 34, тогда в соответствии с матрицей замен символов я шифрую его символом с индексом 82, то есть символом ? Следующие символы открытого текста меняю по тому же алгоритму:

- пробел имеет индекс 94 => 4, то есть => e
- h имеет индекс 7 => 66, то есть => %
- a имеет индекс 0 => 58, то есть => 6
- v имеет индекс 21 => 55, то есть => 3
- e имеет индекс 4 => 92, то есть => }
- пробел имеет индекс 94 => 4, то есть => e
- a имеет индекс 0 => 58, то есть => 6
- пробел имеет индекс 94 => 4, то есть => e
- D имеет индекс 29 => 52, то есть => 0
- r имеет индекс 17 => 6, то есть => g
- e имеет индекс 4 => 92, то есть => }
- a имеет индекс 0 => 58, то есть => 6
- m имеет индекс 12 => 61, то есть => 9
- ! имеет индекс 62 => 34, то есть => I

Сложив все символы в строку получаем **'?e%63}e6e0g}69I'**, что соответствует программно зашифрованной строке.

Расшифрование производится в соответствии с вторым вектором матрицы замены символов. Символ ? в алфавите имеет идекс 82. В первом векторе матрицы по индексу 82 размещена цифра 34, то есть смвол I. Оставшие символы расшифровываются по тому же алгоритму:

- e имеет индекс 4 => 94, то есть => пробел
- % имеет индекс 66 => 7, то есть => h
- 6 имеет индекс 58 => 0, то есть => a
- 3 имеет индекс 55 => 21, то есть => v
- } имеет индекс 92 => 4, то есть => e
- e имеет индекс 4 => 94, то есть => пробел
- 6 имеет индекс 58 => 0, то есть => a
- e имеет индекс 4 => 94, то есть => пробел
- 0 имеет индекс 52 => 29, то есть => D
- g имеет индекс 6 => 17, то есть => r
- } имеет индекс 92 => 4, то есть => e
- 6 имеет индекс 58 => 0, то есть => a
- 9 имеет индекс 61 => 12, то есть => m
- I имеет индекс 34 => 62, то есть => !

Сложив все символы в строку получить **I have a Dream!**

Часть 2. Аффинный шифр

2.1 Программное зашифрование / расшифрование

In [12]:
```
# Программа аффинного шифра реализована
# в модуле simple_cipher классе AffineCipher

from simple_cipher import AffineCipher

# Объявлю объект класса и выведу на экран его атрибуты

encryptor = AffineCipher()

# Алфавит шифрования класс AffineCipher унаследовал от SimpleCipher
# как впрочем и все остальные его атрибуты

encryptor.alphabet
```

Out[12]:
```
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~ '
```

In [13]:
```
# В том числе и модуль
encryptor.module
```

Out[13]: 95

In [14]:
```
# А вот ключ шифрования тут другой
# В аффинном шифровании ключ - это коэффициенты a, b
# необходимые для аффинного преобразрования ax + b,
# где x - это закодированный символ, например его
# порядковый номер в принятом алфавите

# Так как ключ не был передан классу при его объявлении
# то он был автоматически сгененерирован.

encryptor.secret_key
```

Out[14]: (56, 74)

In [15]:
```
# Также ключ шифрования можно передать снаружи

encryptor = AffineCipher((62, 74))
encryptor.secret_key
```

Out[15]: (62, 74)

In [16]:
```
# Попробуем зашифровать какой-то текст
# Определяем еще один текст для шифрования

open_text = "Dum spiro spero!"

# Зашифрую на тех же ключах
encrypted_text = encryptor.encrypt(open_text)
encrypted_text
```

Out[16]: '&<6mY2a@^mY2L@^x'

In [17]:
```
# Расшифруем
decrypted_text = encryptor.decrypt(encrypted_text)
print(decrypted_text)
bool(decrypted_text == open_text)
```

```
Dum spiro spero!
```

Out[17]: True

2.2 Зашифрование / расшифрование вручную

Дано:

- Открытый текст ***"Dum spiro spero!"***
- Алфиавит

  `abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~`

- Ключ шифрования в виде пары чисел

```
(62, 74)
```

- Модуль алгоритма 95
- Таблица индексов символов алфавита

```
{'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4, 'f': 5, 'g': 6, 'h': 7, 'i': 8, 'j': 9, 'k': 10, 'l': 11,
'm': 12, 'n': 13, 'o': 14, 'p': 15, 'q': 16, 'r': 17, 's': 18, 't': 19, 'u': 20, 'v': 21, 'w': 22,
'x': 23, 'y': 24, 'z': 25, 'A': 26, 'B': 27, 'C': 28, 'D': 29, 'E': 30, 'F': 31, 'G': 32, 'H': 33,
'I': 34, 'J': 35, 'K': 36, 'L': 37, 'M': 38, 'N': 39, 'O': 40, 'P': 41, 'Q': 42, 'R': 43, 'S': 44,
'T': 45, 'U': 46, 'V': 47, 'W': 48, 'X': 49, 'Y': 50, 'Z': 51, '0': 52, '1': 53, '2': 54, '3': 55,
'4': 56, '5': 57, '6': 58, '7': 59, '8': 60, '9': 61, '!': 62, '"': 63, '#': 64, '$': 65, '%': 66,
'&': 67, "'": 68, '(': 69, ')': 70, '*': 71, '+': 72, ',': 73, '-': 74, '.': 75, '/': 76, ':': 77,
';': 78, '<': 79, '=': 80, '>': 81, '?': 82, '@': 83, '[': 84, '\\': 85, ']': 86, '^': 87, '_': 88,
'`': 89, '{': 90, '|': 91, '}': 92, '~': 93, ' ': 94}
```

Аффинный шифр с помощью коэффициентов a и b меняет выбирает новый символ из того же словаря для зашифрования:

i - индекс символа из открытого текста

$\hat{i} = (ai + b)$ % m - индекс символа шифртекста

Тогда, для зашифрования требуется выполнить следующие операции над каждым сиволом:

- D - индекс символа 29 => $(62 * 29 + 74) mod 95$ => 67 => &
- u - индекс символа 20 => $(62 * 20 + 74) mod 95$ => 79 => <
- m - индекс символа 12 => $(62 * 12 + 74) mod 95$ => 58 => 6

- пробел - индекс символа 94 => $(62 * 94 + 74) mod 95$ => 12 => m
- s - индекс символа 18 => $(62 * 18 + 74) mod 95$ => 50 => Y
- p - индекс символа 15 => $(62 * 15 + 74) mod 95$ => 54 => 2
- i - индекс символа 8 => $(62 * 8 + 74) mod 95$ => 0 => a
- r - индекс символа 17 => $(62 * 17 + 74) mod 95$ => 83 => @
- o - индекс символа 14 => $(62 * 14 + 74) mod 95$ => 87 => ^
- пробел - индекс символа 94 => $(62 * 94 + 74) mod 95$ => 12 => m
- s - индекс символа 18 => $(62 * 18 + 74) mod 95$ => 50 => Y
- p - индекс символа 15 => $(62 * 15 + 74) mod 95$ => 54 => 2
- e - индекс символа 4 => $(62 * 4 + 74) mod 95$ => 37 => L
- r - индекс символа 17 => $(62 * 17 + 74) mod 95$ => 83 => @
- o - индекс символа 14 => $(62 * 14 + 74) mod 95$ => 87 => ^
- ! - индекс символа 62 => $(62 * 62 + 74) mod 95$ => 23 => x

Собрав все символы в строку получим шифртекст **'&<6mY2a@^mY2L@^x'**

Для расшифрования шифртекста требуется найти элемент обратный a (a - должно быть обратимым по модулю алгоритма)

$a^{-1} = 23$, а затем вычислить индекс исходного элемента открытого текста выполнив операции обратные зашифрованию

$i = (\hat{i} - b) * a^{-1}$ % m - индекс символа шифртекста

Тогда, для расшифрования требуется выполнить следующие операции над каждым сиволом:

- & => имеет индекс 67 => (67 - 74) * 23 mod 95 = 29 => D
- < => имеет индекс 79 => (79 - 74) * 23 mod 95 = 20 => u
- 6 => имеет индекс 58 => (58 - 74) * 23 mod 95 = 12 => m
- m => имеет индекс 12 => (12 - 74) * 23 mod 95 = 94 => пробел
- Y => имеет индекс 50 => (50 - 74) * 23 mod 95 = 18 => s
- 2 => имеет индекс 54 => (54 - 74) * 23 mod 95 = 15 => p
- a => имеет индекс 0 => (0 - 74) * 23 mod 95 = 8 => i
- @ => имеет индекс 83 => (83 - 74) * 23 mod 95 = 17 => r
- ^ => имеет индекс 87 => (87 - 74) * 23 mod 95 = 14 => o
- m => имеет индекс 12 => (12 - 74) * 23 mod 95 = 94 => пробел
- Y => имеет индекс 50 => (50 - 74) * 23 mod 95 = 18 => s
- 2 => имеет индекс 54 => (54 - 74) * 23 mod 95 = 15 => p
- L => имеет индекс 37 => (37 - 74) * 23 mod 95 = 4 => e
- @ => имеет индекс 83 => (83 - 74) * 23 mod 95 = 17 => r
- ^ => имеет индекс 87 => (87 - 74) * 23 mod 95 = 14 => o
- x => имеет индекс 23 => (23 - 74) * 23 mod 95 = 62 => !

Сложив все символы в строку получим **Dum spiro spero!**

Часть 3. Аффинный рекуррентный шифр

3.1 Программное зашифрование / расшифрование

In [18]:
```
# Программа аффинного рекуррентного шифра реализована
# в модуле simple_cipher классе AffineRecurrentCipher

from simple_cipher import AffineRecurrentCipher

# Объявлю объект класса и выведу на экран его атрибуты

encryptor = AffineRecurrentCipher()

# Алфавит шифрования класс AffineRecurrentCipher унаследовал от AffineCipher
# как впрочем и все остальные его атрибуты

encryptor.alphabet
```

Out[18]:
```
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~ '
```

In [19]:
```
# В том числе и модуль
encryptor.module
```

Out[19]:
```
95
```

In [20]:
```
# Но для зашифрования и расшифрования аффинный рекуррентный шифр
# использует другие ключи. У него их изначально два, но на
# каждом шаге шифрования (начиная со второго блока) генерируется
# новый на основе двух предыдущих

encryptor.secret_keys
```

Out[20]:
```
[(61, 14), (67, 93)]
```

In [21]:
```
# Чтобы не пользоваться ключами по-умолчанию алгоритму можно
# передать свои ключи, например вот так:

encryptor = AffineRecurrentCipher((39, 54), (18, 46))
encryptor.secret_keys
```

Out[21]:
```
[(39, 54), (18, 46)]
```

In [22]:
```
# Попробуем зашифровать какой-то текст
# Определяем еще один текст для шифрования

open_text = "This is the way!"

# Зашифрую на тех же ключах
encrypted_text = encryptor.encrypt(open_text)
encrypted_text
```

Out[22]:
```
'e:q(txk!|TX}H#GE'
```

In [23]:
```
# Расшифрую и сравню с исходным текстом
decrypted_text = encryptor.decrypt(encrypted_text)
print(decrypted_text)
bool(decrypted_text == open_text)
```

```
This is the way!
```

Out[23]:
```
True
```

3.2 Зашифрование / расшифрование вручную

Дано:

- Открытый текст ***"This is the way!"***
- Алфиавит

```
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~
```

- Ключи шифрования в виде двух пар чисел

```
[(39, 54), (18, 46)]
```

- Модуль алгоритма 95
- Таблица индексов символов алфавита

```
{'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4, 'f': 5, 'g': 6, 'h': 7, 'i': 8, 'j': 9, 'k': 10, 'l': 11,
'm': 12, 'n': 13, 'o': 14, 'p': 15, 'q': 16, 'r': 17, 's': 18, 't': 19, 'u': 20, 'v': 21, 'w': 22,
'x': 23, 'y': 24, 'z': 25, 'A': 26, 'B': 27, 'C': 28, 'D': 29, 'E': 30, 'F': 31, 'G': 32, 'H': 33,
'I': 34, 'J': 35, 'K': 36, 'L': 37, 'M': 38, 'N': 39, 'O': 40, 'P': 41, 'Q': 42, 'R': 43, 'S': 44,
'T': 45, 'U': 46, 'V': 47, 'W': 48, 'X': 49, 'Y': 50, 'Z': 51, '0': 52, '1': 53, '2': 54, '3': 55,
'4': 56, '5': 57, '6': 58, '7': 59, '8': 60, '9': 61, '!': 62, '"': 63, '#': 64, '$': 65, '%': 66,
'&': 67, "'": 68, '(': 69, ')': 70, '*': 71, '+': 72, ',': 73, '-': 74, '.': 75, '/': 76, ':': 77,
';': 78, '<': 79, '=': 80, '>': 81, '?': 82, '@': 83, '[': 84, '\\': 85, ']': 86, '^': 87, '_': 88,
'`': 89, '{': 90, '|': 91, '}': 92, '~': 93, ' ': 94}
```

Также как и в обычном аффинном шифре в аффинном рекуррентном шифре для шифрования используется формула

$\hat{i} = (ai + b)$ % m, однако коэффициента a и b на каждом символе разные.

Пересчет коэффициентов выполняется по формуле:

$a_i = a_{i-1} * a_{i-2}$ % m

$b_i = b_{i-1} + b_{i-2}$ % m

Зашифруем строку "This is the way!"

- T => i = 45, a = 39, b = 54 => $\hat{i}$ = (39*45+54)%95 = 4 => e
- h => i = 7, a = 18, b = 46 => $\hat{i}$ = (18*7+46)%95 = 77 => :
- i => i = 8, a = 37, b = 5 => $\hat{i}$ = (37*8+5)%95 = 16 => q
- s => i = 18, a = 1, b = 51 => $\hat{i}$ = (18+51)%95 = 69 => (
- пробел => i = 94, a = 37, b = 56 => $\hat{i}$ = (94*37+56)%95 = 19 => t
- i => i = 8, a = 37, b = 12 => $\hat{i}$ = (8*37+42)%95 = 23 => x
- s => i = 18, a = 39, b = 68 => $\hat{i}$ = (18+23)%95 = 10 => k
- пробел => i = 94, a = 18, b = 80 => $\hat{i}$ = (94+65)%95 = 62 => !
- t => i = 19, a = 37, b = 53 => $\hat{i}$ = (19+88)%95 = 91 => |
- h => i = 7, a = 1, b = 38 => $\hat{i}$ = (7+38)%95 = 45 => T
- e => i = 4, a = 37, b = 91 => $\hat{i}$ = (4*37+91)%95 = 49 => X
- пробел => i = 94, a = 37, b = 34 => $\hat{i}$ = (94*37+34)%95 = 92 => }

- w => i = 22, a = 39, b = 30 => $\hat{i}$ = (22*39+30)%95 = 33 => H
- a => i = 0, a = 18, b = 64 => $\hat{i}$ = (0+64)%95 = 64 => #
- y => i = 24, a = 37, b = 94 => $\hat{i}$ = (24*37+94)%95 = 32 => G
- ! => i = 62, a = 1, b = 63 => $\hat{i}$ = (62+63)%95 = 30 => E

Сложим все символы в одну строку и получим 'e:q(txk!|TX}H#GE'

Для расшифровки, так же как и в аффинном шифре требуется вычислить a^{-1}, но поскольку a на каждом шаге меняется то и вычислять его нужно каждый раз заново. Кэффициент b такой же как при вычислениях выше. Расшифрование производится по формуле:

$$i = (\hat{i} - b) * a^{-1}$$

- e => $\hat{i}$ = 4 , a^{-1} = 39 => (4-54)*39 % 95 = 45 => T
- : => $\hat{i}$ = 77, a^{-1} = 37 => (77-46)*37 % 95 = 7 => h
- q => $\hat{i}$ = 16, a^{-1} = 18 => (16-5)*18 % 95 = 8 => i
- (=> $\hat{i}$ = 69, a^{-1} = 1 => (69 - 51)*1 % 95 = 18 => s
- t => $\hat{i}$ = 19, a^{-1} = 18 => (19 - 56)*18 % 95 = 94 => пробел
- x => $\hat{i}$ = 23, a^{-1} = 18 => (23 - 12)*18 % 95 = 8 => i
- k => $\hat{i}$ = 10, a^{-1} = 39 => (10 - 68)*39 % 95 = 18 => s
- ! => $\hat{i}$ = 62, a^{-1} = 37 => (62 - 80)*37 % 95 = 94 => пробел
- | => $\hat{i}$ = 91, a^{-1} = 18 => (91 - 53)*18 % 95 = 19 => t
- T => $\hat{i}$ = 45, a^{-1} = 1 => (45 - 38)*1 % 95 = 7 => h
- X => $\hat{i}$ = 49, a^{-1} = 18 => (49 - 91)*18 % 95 = 4 => e
- } => $\hat{i}$ = 92, a^{-1} = 18 => (92 - 34)*18 % 95 = 94 => пробел
- H => $\hat{i}$ = 33, a^{-1} = 39 => (33 - 30)*39 % 95 = 22 => w
- '#' => $\hat{i}$ = 64, a^{-1} = 37 => (64 - 64)*37 % 95 = 0 => a
- G => $\hat{i}$ = 32, a^{-1} = 18 => (32 - 94)*18 % 95 = 24 => y
- E => $\hat{i}$ = 30, a^{-1} = 1 => (30 - 63)*1 % 95 = 62 => !

Сложим всё в одну строку и получим **This is the way!**