Оглавление

[**Алгоритм SPN** 2](#_Toc55191478)

[**Замена бит в блоке** 3](#_Toc55191479)

[**Перестановка бит в блоке** 6](#_Toc55191480)

[**Алгоритм генерации подключей для шифрования** 8](#_Toc55191481)

[**Алгоритм шифрования** 9](#_Toc55191482)

[**Задание 1** 10](#_Toc55191483)

[**Задание 2** 11](#_Toc55191484)

[**Задание 3** 12](#_Toc55191485)

[**Алгоритм расшифрования** 14](#_Toc55191486)

[**Алгоритм генерации подключей для расшифрования** 16](#_Toc55191487)

[**Задание 4** 16](#_Toc55191488)

[**Задание 5** 17](#_Toc55191489)

[**Задание 6** 18](#_Toc55191490)

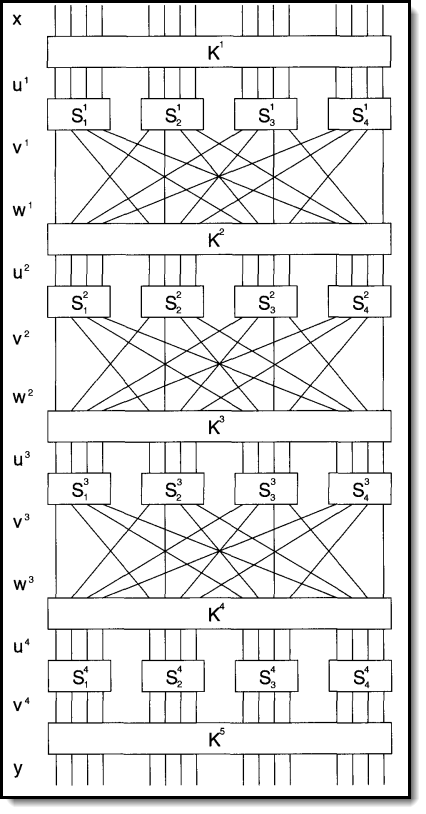
[**Задание 7** 18](#_Toc55191491)

[**Задание 8** 19](#_Toc55191492)

[Литература 20](#_Toc55191493)

**Алгоритм SPN**

Рассмотрим алгоритм шифрования, построенный на основе сети SPN, структура которого показана на рис. 1. Здесь  - 16-ти битовый блок открытого (исходного) сообщения,  - 16-ти битовый блок закрытого (зашифрованного) сообщения.



1. – Структура алгоритма шифрования, построенного на основе сети SPN [1, 2, 3]

В основе алгоритма – последовательное применение двух основных преобразований: замены 



и перестановки

,

где  - размер блока. В алгоритме, представленном на рис. 1, =4, =4.

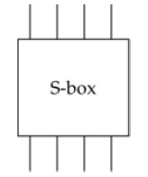
**Замена бит в блоке**

Преобразование  можно задать в виде таблицы, где первая строка задает вход (), а вторая строка – выход (). Табл.1 задает используемое в данном алгоритме преобразование .

Таблица 1 - Замена

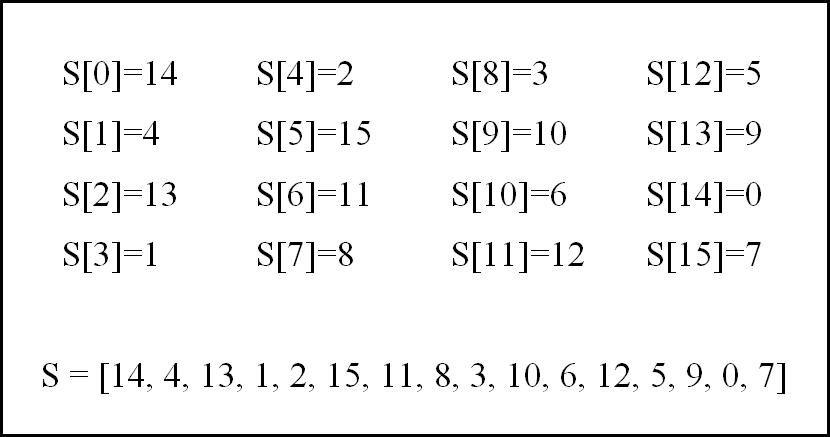
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Вход | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Выход | 14 | 4 | 13 | 1 | 2 | 15 | 11 | 8 | 3 | 10 | 6 | 12 | 5 | 9 | 0 | 7 |

На схемах, описывающих алгоритмы шифрования, преобразование замены принято обозначать, как показано на рис.2.



1. – Графическое обозначение замены

В частности, в схеме алгоритма на рис. 1 операции замены обозначены именно таким образом – в виде S-блоков замены. На вход блока замены поступает 4-х битовое значение, на выходе блока замены – измененное в соответствии с табл.1 4-х битовое значение. Табл.1 можно описать в виде массива (рис.3).



1. – Реализация блока замены

Для выполнения замены  необходимо выполнить следующие шаги:

1. -битовый блок разбить на  -битовых подблоков. Такое разбиение для -битового блока  можно записать таким образом

,

где 

2. Применить преобразование  над каждым подблоком:



3. Объединить подблоки в один -битовый вход

.

Пример выполнения замены для 16-битового блока приведен на рис.4.

|  |
| --- |
| Путь . Тогда 16-ти битовый блок  разбивается на 4 подблока , где  ,  ,  ,  .  Для блока  получим 4 подблока:  .  ,  ,  .  После применения преобразования  (табл.1) над каждым из подблоков получим  ,  ,  ,  .  Таким образом, результатом применения преобразования замены над блоком будет блок . |

1. – Пример выполнения операции замены

**Перестановка бит в блоке**

Преобразование  задает перестановку бит внутри блока. Данное преобразование удобно задать в виде таблицы, в которой в первой строке (вход z) заданы порядковые номера  бит блока (самый младший бит в блоке имеет номер 0), а во второй строке - выход () – результат перестановки бит внутри блока, т.е. на -ю позицию ставится  бит блока (табл.2).

Таблица 2 - Перестановка

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| вход | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Выход | 15 | 11 | 7 | 3 | 14 | 10 | 6 | 2 | 13 | 9 | 5 | 1 | 12 | 8 | 4 | 0 |

Указанный способ перестановки реализован с помощью метода pbox() (рис.5).

|  |
| --- |
| def pbox(self, x):  y = 0  for i in range(len(self.p)):  if (x & (1 << i)) != 0:  y ^= (1 << self.p[i])  return y |



Например, для блока  результатом применения преобразования  (табл.2) будет блок (рис.).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| вход | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  | 1 |  |  |  | 1 |  | 1 | 1 | 1 |  | 1 |  |  |  | 1 |
| выход | 15 | 11 | 7 | 3 | 14 | 10 | 6 | 2 | 13 | 9 | 5 | 1 | 12 | 8 | 4 | 0 |
|  |  | 1 |  | 1 | 1 | 1 |  |  |  |  |  |  | 1 | 1 | 1 |

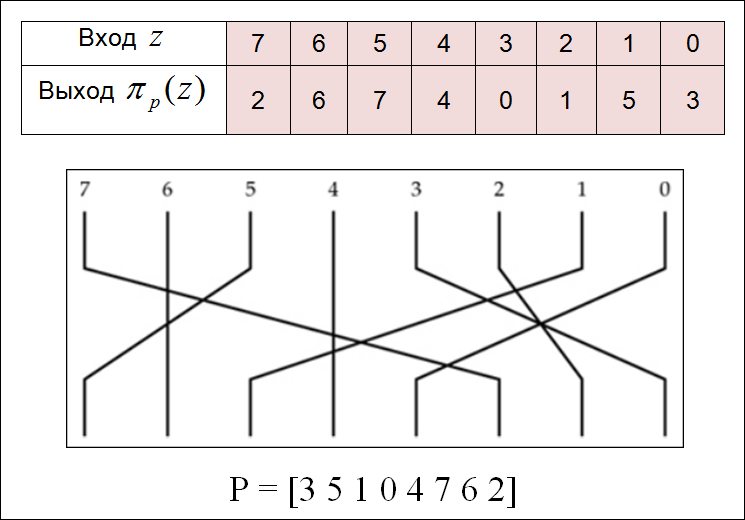


На рис.6 пустая ячейка в таблице означает нуль. Имеет смысл переставлять только единицы, как это реализовано в методе pbox() (рис.5). Закрашенные одинаковым цветом ячейки таблицы показывают перемещение соответствующих единиц в результате перестановки.

Перестановку (табл.2) удобно задать в виде массива:

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

В схеме на рис.1 перестановка показана традиционным графическим способом как в примере на рис.7.





Необходимо обеспечить взаимно однозначное соответствие между входом  и выходом . Также, для задания перестановки в программе требуется существенно меньше памяти. Так, для задания 16-ти битовой перестановки требуется массив из 16 чисел, тогда как для задания 16-ти битовой замены требуется хранить массив из 65536 чисел.

**Алгоритм генерации подключей для шифрования**

Составной частью алгоритма является описание процедуры получения раундовых ключей – так называемой процедуры генерации подключей. Для рассматриваемого алгоритма шифрования процедура генерации подключей заключается в следующем: все пять подключей получаются последовательным выбором 16 бит из 32 битного ключа по следующему правилу. Ключ  () состоит из 16 последовательных бит ключа , начиная с . Например, для ключа = 982832703 (0011 1010 1001 0100 1101 0110 0011 1111) в результате применения процедуры генерации подключей (расширения ключа) получены следующие раундовые ключи (рис.).

|  |
| --- |
| = 0011 1010 1001 0100  =1010 1001 0100 1101  =1001 0100 1101 0110  =0100 1101 0110 0011  =1101 0110 0011 1111 |

1. – Раундовые ключи (подключи)

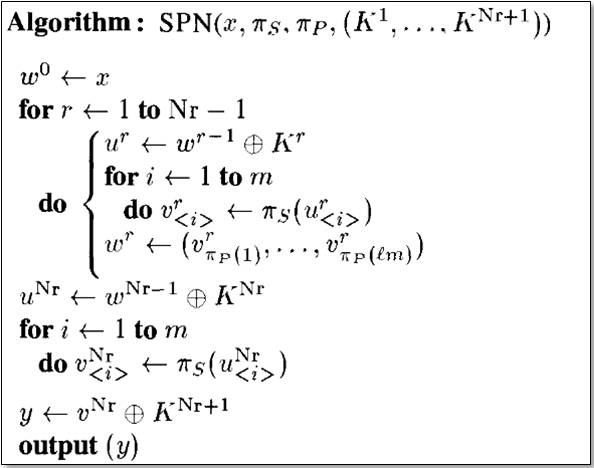
Алгоритм реализован с помощью метода round\_keys() (рис.9).

|  |
| --- |
| def round\_keys(self, k):  rk = []  rk.append((k >> 16) & (2\*\*16-1))  rk.append((k >> 12) & (2\*\*16-1))  rk.append((k >> 8) & (2\*\*16-1))  rk.append((k >> 4) & (2\*\*16-1))  rk.append(k & (2\*\*16-1))  return rk |



**Алгоритм шифрования**

Псевдокод алгоритма шифрования приведен на рис.10.  - количество раундов шифрования.



1. – Псевдокод алгоритма шифрования

Для открытого 16-ти битового блока  и ключа  последовательное применение алгоритма (рис.) дает результаты, приведенные на рис..

|  |
| --- |
| Раунд 1        Раунд 2        Раунд 3        Раунд 4    = 0110101011101001 |



В последнем раунде отсутствует перестановка бит после операции замены и выполняется дополнительное сложение по модулю 2 с пятым подключом. Так сделано для того, чтобы использовать ту же самую схему (рис.1) и для расшифрования данных.

**Задание 1**

а) Используя раундовые ключи как на рис.8 выполнить один раунд шифрования 16-ти битового значения, сформированного из первых букв фамилии и имени. Результат представить, как на рис.11.

б) В файле spn1.py содержится реализация алгоритма шифрования. Пояснить, что делает функция demux():

|  |
| --- |
| import spn1  e = spn1.SPN1()  x = 15324  print('x={}'.format(bin(x)[2:].zfill(16)))  y = e.demux(x)  print('y={}'.format(y)) |

в) Пояснить, что делает функция mux():

|  |
| --- |
| Import spn1  e = spn1.SPN1()  x = [9, 11, 4, 2]  y = e.mux(x)  print('y={}'.format(bin(y)[2:].zfill(16))) |

г) Зашифровать блок данных, который использовался для выполнения задания из п. а). На рис.12 представлен код, который можно использовать для выполнения задания. Ключ шифрования = 982832703. Проверьте, совпадает ли результат с результатом, полученным при выполнении задания из п. а).

Распечатать значения раундовых ключей шифрования как на рис.8. Распечатать значения блока после каждого шага шифрования, как на рис.11.

|  |
| --- |
| import spn1  x = 15324 % здесь индивидуальное значение  print('x={}'.format(bin(x)[2:].zfill(16)))  k = 982832703  print('k={}'.format(bin(k)[2:].zfill(32)))  e = spn1.SPN1()  rk = e.round\_keys(k)  y = e.encrypt(x, rk, rounds=4)  print('y={}'.format(bin(y)[2:].zfill(16))) |



**Задание 2**

Написать функцию encrypt\_data(self, data, key, rounds), где data – список чисел (данные, прочитанные из файла), key – ключ шифра, rounds – количество раундов.

В этой функции надо сформировать список раундовых ключей шифрования и для каждого числа (16 бит) в списке data вызывать функцию encrypt. Функция возвращает список зашифрованных данных (рис.13).

|  |
| --- |
| data = [15324, 3453, 34, 12533]  k = 734533245  e = spn1.SPN1()  cypher\_data = e.encrypt\_data(data, key=k, rounds=4)  print('cypher\_data={}'.format(cypher\_data))  cypher\_data=[8144, 26070, 3827, 38912] |



**Задание 3**

а) Добавить в класс SPN1 метод asbox(), который выполняет обратную замену:

|  |
| --- |
| import spn1  e = spn1.SPN1()  x = 9  sx = e.sbox(x)  print('x={}--->s[{}]={}'.format(x, x, sx))  x\_ = e.asbox(sx)  print('as[{}]={}'.format(sx, x\_)) |

Можно использовать метод списка index().

б) Обратная перестановка  реализована с помощью метода apbox() (рис.14).

|  |
| --- |
| def apbox(self, x):  y = 0  for i in range(len(self.p)):  if (x & (1 << self.p[i])) != 0:  y ^= (1 << i)  return y |

1. – Обратная перестановка

Проверьте корректность выполнения обратной перестановки для p = [2, 5, 6, 8, 4, 14, 0, 7, 11, 10, 12, 1, 15, 9, 3, 13]

|  |
| --- |
| import spn1  e = spn1.SPN1()  x = int('0010011010110111', 2)  px = e.pbox(x)  print('x={}--->px={}'.format(bin(x)[2:].zfill(16), bin(px)[2:].zfill(16)))  x\_ = e.apbox(px)  print('px={}--->x\_={}'.format(bin(px)[2:].zfill(16), bin(x\_)[2:].zfill(16))) |

в) Проверьте выполнение равенства , например, для  и .

**Алгоритм расшифрования**

Для расшифрования можно использовать ту же схему, что и для шифрования данных (рис.1). В самом деле, посмотрим, как можно выполнить расшифрование по этой схеме, выполняя обратные преобразования, двигаясь снизу вверх по схеме, т.е. из  получить (рис.15).

|  |  |
| --- | --- |
| C:\Users\836D~1\AppData\Local\Temp\SNAGHTML2aefbf8.PNG |  |



Можно заметить, что получились выражения, такие же, как и при шифровании с учетом того, что операция замены меняется на обратную, перестановка меняется на обратную перестановку и используются другие ключи (рис.16).

|  |
| --- |
|  |

1. – Расшифрование по схеме на рис.1

Таким образом, для расшифрования будет использоваться та же схема, что и для шифрования (рис.1), в которой операции замены и перестановки заменены на их обратные и вместо подключей шифрования будут использоваться подключи расшифрования, алгоритм вычисления которых непосредственно следует из выражений на рис.15, 16.

**Алгоритм генерации подключей для расшифрования**

Раундовые ключи для расшифрования рассчитываются из раундовых ключей шифрования  по следующим формулам:

 (1)

Для ключа key = 0011 1010 1001 0100 1101 0110 0011 1111 рассчитанные значения раундовых ключей для расшифрования приведены на рис.17.

|  |
| --- |
| K0=0011 1010 1001 0100, L0=1101 0110 0011 1111  K1=1010 1001 0100 1101, L1=0100 1110 0011 0101  K2=1001 0100 1101 0110, L2=1010 0111 0001 1010  K3=0100 1101 0110 0011, L3=1101 0011 1000 0101  K4=1101 0110 0011 1111, L4=0011 1010 1001 0100 |

1. – Подключи шифрования и расшифрования

**Задание 4**

Написать метод round\_keys\_to\_decrypt(self, key) , где key – ключ шифрования (рис.18). Функция формирует список раундовых ключей для расшифрования по формуле (1).

|  |
| --- |
| def round\_keys\_to\_decrypt(self, key):  K = self.round\_keys(key)  L = [ ]  % код  return L |



Для =734533245 результат работы функции приведен на рис.19.

|  |
| --- |
| L0=0001011001111101  L1=1000001100110101  L2=1100100100010010  L3=1110010010001001  L4=0010101111001000 |



**Задание 5**

Написать метод decrypt(self, x, rl, rounds), который выполняет расшифрование одного блока данных (числа x). rl-список ключей для расшифрования, rounds-количество раундов (в данном случае - 4). Структурно метод совпадает с методом шифрования encrypt (рис.20), т.к. используется одна и та же схема (рис.1).

|  |
| --- |
| # Шифруем одно число  def encrypt(self, x, rk, rounds):  for i in range(rounds-1):  x = self.round(x, rk[i])  x = self.last\_round(x, rk[rounds-1], rk[rounds])  return x |



Отличие заключается в том, что вместо round и last\_round надо вызывать другие функции, т.к. обработка внутри раундов меняется – используются обратные перестановки и замены. Следовательно, надо еще добавить методы round\_decrypt и last\_round\_decrypt.

Пример вызова метода приведен на рис..

|  |
| --- |
| x = 9911  k = 982832703  print('x={}'.format(bin(x)[2:].zfill(16)))  rk = e.round\_keys(k)  y = e.encrypt(x, rk, rounds=4)  lk = e.round\_keys\_to\_decrypt(k)  x\_ = e.decrypt(y, lk, rounds=4)  print('y={}'.format(bin(y)[2:].zfill(16)))  print('x\_={}'.format(bin(x\_)[2:].zfill(16)))  x=0010011010110111  y=1011110011010110  x\_=0010011010110111 |



**Задание 6**

Написать функцию decrypt\_data(self, data, key, rounds), где data –список чисел (данные, прочитанные из зашифрованного файла), key – ключ шифра, rounds – количество раундов. В этой функции надо сформировать список раундовых ключей расшифрования и для каждого числа (16 бит) в списке data вызвать функцию decrypt. Функция возвращает список расшифрованных данных (рис.22).

|  |
| --- |
| x = [9911, 12432, 456, 21]  k = 982832703  print('x={}'.format(x))  y = e.encrypt\_data(x, k, rounds=4)  x\_ = e.decrypt\_data(y, k, rounds=4)  print('y={}'.format(y))  print('x\_={}'.format(x\_))  x=[9911, 12432, 456, 21]  y=[48342, 41317, 8756, 23451]  x\_=[9911, 12432, 456, 21] |



**Задание 7**

Зашифровать и расшифровать содержимое файла (‘123.txt’) с помощью функций encrypt\_data и decrypt\_data. Для получения содержимого файла в виде списка чисел использовать функцию read\_data\_2byte. Для записи функции в файл использовать функцию write\_data\_2byte.

Убедиться, что расшифрованный после шифрования файл совпадает с исходным (рис.23).

|  |
| --- |
| from read\_write\_file import read\_data\_2byte as read2  from read\_write\_file import write\_data\_2byte as write2  e = spn1.SPN1()  # шифрование  data = read2('123.txt')  cypher\_data = e.encrypt\_data(data, key=452342216, rounds=4)  write2('123\_encrypt.txt', cypher\_data)  # расшифрование  data = read2('123\_encrypt.txt')  decrypt\_data = e.decrypt\_data(data, key=452342216, rounds=4)  write2('123\_decrypt.txt', decrypt\_data) |



**Задание 8**

Расшифровать файл d5\_spn\_c\_all.bmp – зашифрованное шифром на основе сети SPN изображение в формате bmp. Ключ равен 34523456231.

Полученное изображение в формате bmp зашифровать. Сохранить в файле следующие данные: первые 50 байт – исходные (незашифрованные) данные, все последующие байты – зашифрованные.

# Литература

[1] Бабенко Л.К., Ищукова Е.А. "Современные алгоритмы блочного шифрования и методы их анализа. Учебное пособие для вузов",  Гелиос АРВ, 2006, 376 с.

[2] Douglas R. Stinson. Cryptography: Theory and Practice, Third Edition (Discrete Mathematics and Its Applications), p.616, 2005.

[3] Christopher Swenson. Modern Cryptanalysis: Techniques for Advanced Code Breaking, Wiley Publishing, 2008

[4] Heys, H. (2002). A tutorial on linear and differential cryptanalysis. Cryptologia, 26(3), pp. 189-221.