Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

ОТЧЁТ

по лабораторной работе №1

По дисциплине «Методы защиты информации»

Выполнил:

Студент гр. 853505

Лазарева Е.В.

Проверил:

Олисейчик В.В.

Минск 2021

**1. Постановка задачи**

Необходимо реализовать программные средства шифрования и дешифрования текстовых файлов при помощи алгоритмов DES и *ГОСТ 28147-89.*

**ЗАДАНИЕ:**

1. Изучить теоретические сведения.
2. Создать программы, читающие данные из файла и шифрующие (дешифрующие) их с помощью с помощью DES (двойной и тройной) и *ГОСТ 28147-89*.

**2. Теоретическая** **сведения**

**Алгоритм шифрования DES.**

**DES** (Data Encryption Standard) — алгоритм для симметричного шифрования, в котором данные шифруются 64-битными блоками и специальным ключом с длиной 56 бит. Алгоритм использует комбинацию нелинейных (S-блоки) и линейных (перестановки E, IP, IP-1) преобразований. Алгоритм преобразует за несколько раундов 64-битный вход в 64-битный выход.

Процесс шифрования состоит из четырех этапов:

1. Выполняется начальная перестановка (IP) 64-битного исходного текста (забеливание), во время которой биты переупорядочиваются в соответствии со стандартной таблицей.
2. Этап состоит из 16 раундов одной и той же функции, которая использует операции сдвига и подстановки.
3. Левая и правая половины выхода последней (16-й) итерации меняются местами.
4. Выполняется перестановка IP-1 результата, полученного на третьем этапе. Перестановка IP-1 инверсна начальной перестановке.

Прямым развитием DES в настоящее время является алгоритм Doube DES (2DES) и Triple DES (3DES). В них шифрование/расшифровка выполняются путём двоекратного и троекратного выполнения алгоритма DES соответственно.

Схема шифрования с помощью алгоритма DES:



Рисунок 1 – Схема шифрования с помощью алгоритма DES.

**Начальная и конечная перестановки.**

Начальная перестановка и ее инверсия определяются стандартной таблицей. Если М – это произвольные 64 бита, то X = IP (M) – это переставленные 64 бита. Если применить обратную функцию перестановки Y = IP-1 (X) = IP-1 (IP(M)), то получится первоначальная последовательность бит.

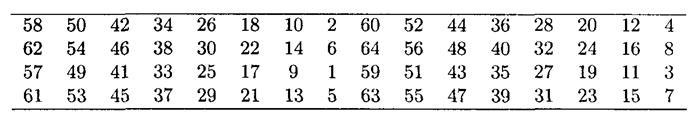


Рисунок 2 - DES. Начальная перестановка

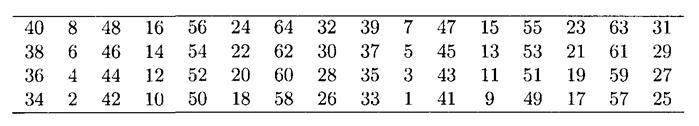


Рисунок 3 - DES. Заключительная перестановка

**Последовательность преобразований отдельного раунда.**

Теперь рассмотрим последовательность преобразований, используемую в каждом *раунде*.

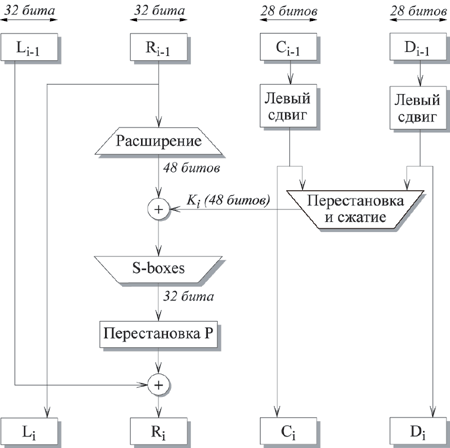


Рисунок 4 - i-ый раунд DES

64-битный входной блок проходит через 16 *раундов*, при этом на каждой итерации получается промежуточное 64-битное значение. Левая и правая части каждого промежуточного значения трактуются как отдельные 32-битные значения, обозначенные *L* и *R*.

Каждую итерацию можно описать следующим образом:

Li = Ri-1

Ri = Li-1 F(Ri-1, Ki) где, – это операцию XOR.

Таким образом, выход левой половины *Li* равен входу правой половины *Ri-1*. Выход правой половины *Ri*является результатом применения операции XOR к *Li-1* и функции *F*, зависящей от *Ri-1* и *Ki*.

Рассмотрим функцию *F* более подробно:

*Ri*, которое подается на вход функции *F*, имеет длину 32 бита. Вначале Ri расширяется до 48 бит, используя таблицу, которая определяет перестановку плюс расширение на 16 бит. Расширение происходит следующим образом. 32 бита разбиваются на группы по 4 бита и затем расширяются до 6 бит, присоединяя крайние биты из двух соседних групп. Например, если часть входного сообщения:

**. . . efgh ijkl mnop . . .**

то в результате расширения получается сообщение:

**. . . defghi hijklm lmnopq . . .**

После этого для полученного 48-битного значения выполняется операция XOR с 48-битным *подключом Ki*. Затем полученное 48-битное значение подается на вход функции подстановки, результатом которой является 32-битное значение.

Подстановка состоит из восьми *S-boxes,* каждый из которых на входе получает 6 бит, а на выходе создает 4 бита. Эти преобразования определяются специальными таблицами. Первый и последний биты входного значения *S-box* определяют номер строки в таблице, средние 4 бита определяют номер столбца. Пересечение строки и столбца определяет 4-битный выход. Например, если входом является 011011, то номер строки равен 01 (строка 1) и номер столбца равен 1101 (столбец 13). Значение в строке 1 и столбце 13 равно 5, т.е. выходом является 0101.

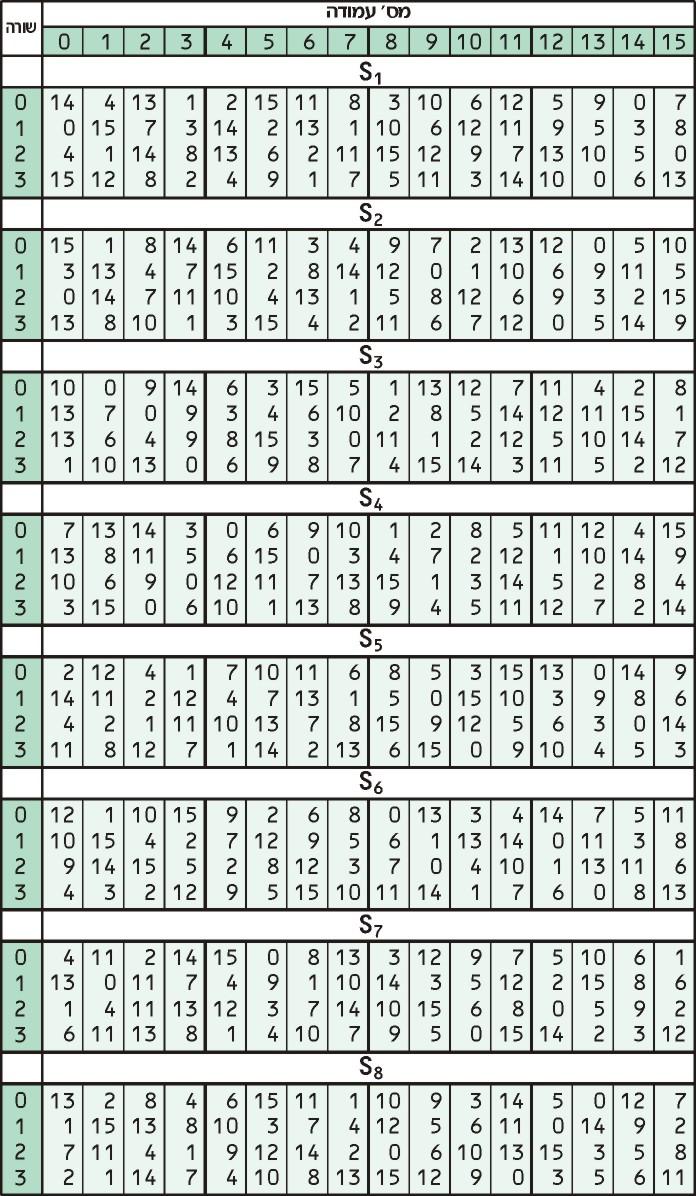


Рисунок 5 - S-boxes

Далее полученное 32-битное значение обрабатывается с помощью перестановки *Р*, целью которой является максимальное переупорядочивание бит, чтобы в следующем *раунде* шифрования с большой вероятностью каждый бит обрабатывался другим *S-box*.

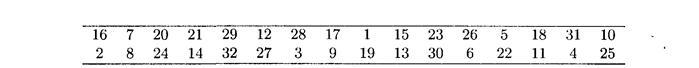


Рисунок 6 - Перестановка с помощью Р-блоков

**Создание подключей.**

Ключ для отдельного *раунда Ki*состоит из 48 бит. Ключи *Ki* получаются по следующему алгоритму. Для 56-битного ключа, используемого на входе алгоритма (если используется 64-битный ключ, то, как видно из рис. 6 убираются биты 64, 56, 48, 40, 32, 16, 8), вначале выполняется перестановка в соответствии с таблицей Permuted Choice 1 (РС-1).

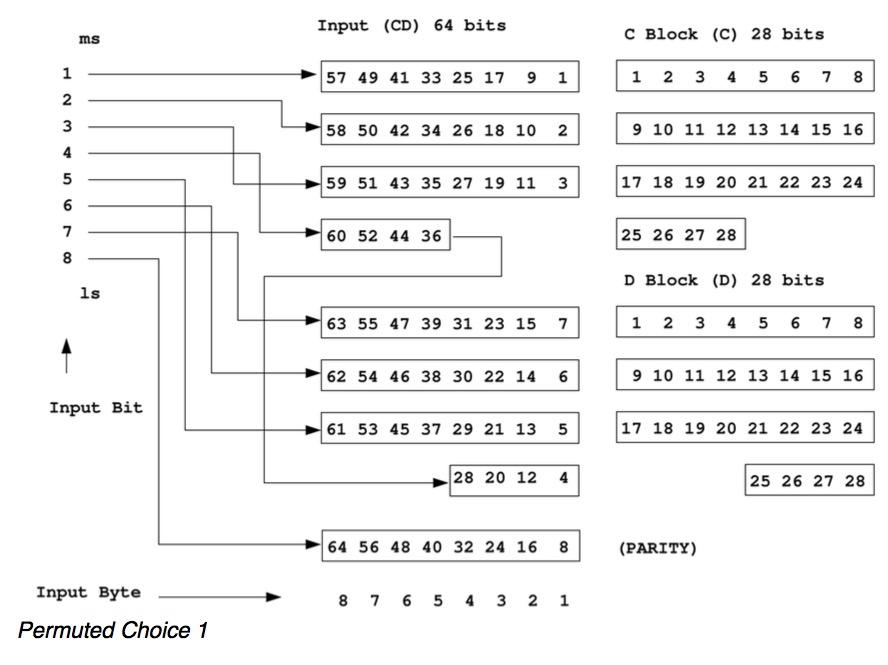


Рисунок 7 - Схема Permuted Choice

Полученный 56-битный ключ разделяется на две 28-битные части, обозначаемые как C0 и D0 соответственно. На каждом *раунде Ci* и *Di*независимо циклически сдвигаются влево на 1 или 2 бита, в зависимости от номера *цикла*.

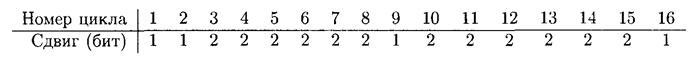


Рисунок 8 - Сдвиг ключа в зависимости от номера цикла

Полученные значения являются входом следующего *раунда*. Они также представляют собой вход в Permuted Choice 2 (РС-2), который создает 48-битное выходное значение, являющееся входом функции *F*(*Ri-1*, *Ki*).

Процесс ***дешифрования*** аналогичен процессу шифрования. На входе алгоритма используется зашифрованный текст, но ключи *Ki* используются в обратной последовательности. *K16* используется на первом *раунде*, *K1* используется на последнем *раунде*.

**Двойной DES.**

Наиболее логичным способом противодействия полному перебору ключа DES выглядит многократное шифрование данных алгоритмом DES с различными ключами. Следующий алгоритм получил название Double DES (двойной DES):

где: — половины двойного ключа алгоритма Double DES, каждая из которых представляет собой обычный 56-битный ключ DES; Е — функция шифрования блока данных обычным алгоритмом DES. Если бы при двойном шифровании DES выполнялось следующее свойство: для любых значений, то двойное шифрование не приводило бы к усилению против полного перебора ключа — всегда нашелся бы такой ключ *k*, *однократное* шифрование которым было бы эквивалентно двукратному шифрованию на ключах, а для нахождения ключа к достаточно было бы перебрать 255 ключей. Double DES действительно удваивает эффективный размер ключа— до 112 битов, а при современном развитии вычислительной техники полный перебор 112-битного ключа невозможен.

**Тройной DES.**

Triple DES (3DES) — симметричный [блочный шифр](https://ru.wikipedia.org/wiki/Блочный_шифр), созданный [Уитфилдом Диффи](https://ru.wikipedia.org/wiki/Диффи,_Уитфилд), [Мартином Хеллманом](https://ru.wikipedia.org/wiki/Хеллман,_Мартин) и Уолтом Тачманном в [1978 году](https://ru.wikipedia.org/wiki/1978_год) на основе алгоритма [DES](https://ru.wikipedia.org/wiki/DES) с целью устранения главного недостатка последнего — малой длины ключа (56 бит), который может быть взломан методом [полного перебора](https://ru.wikipedia.org/wiki/Полный_перебор) ключа. Скорость работы 3DES в 3 раза ниже, чем у DES, но криптостойкость намного выше — время, требуемое для криптоанализа 3DES, может быть в миллиард раз больше, чем время, нужное для вскрытия DES. 3DES используется чаще, чем DES, который легко взламывается при помощи сегодняшних технологий (в [1998 году](https://ru.wikipedia.org/wiki/1998_год) организация [Electronic Frontier Foundation](https://ru.wikipedia.org/wiki/Electronic_Frontier_Foundation), используя специальный компьютер *DES Cracker*, вскрыла DES за 3 дня). 3DES является простым способом устранения недостатков DES. Алгоритм 3DES построен на основе DES, поэтому для его реализации возможно использовать программы, созданные для DES.

Официальное название алгоритма, используемое в стандартах - TDEA или Triple DEA (англ. Triple Data Encryption Algorithm). Однако, термин "3DES" используется более широко поставщиками, пользователями и разработчиками криптосистем.

Существует 2 основных варианта шифрования алгоритмом 3DES: 3-key Triple DES и 2-key Triple DES. Как видно из названия, принципиальное различие этих методов - количество ключей (три и два соответственно). В свою очередь, каждый из этих алгоритмов имеет по 2 разных типа: EEE (encryption-encryption-encryption) и EDE (encryption-decryption-encryption) шифрование.

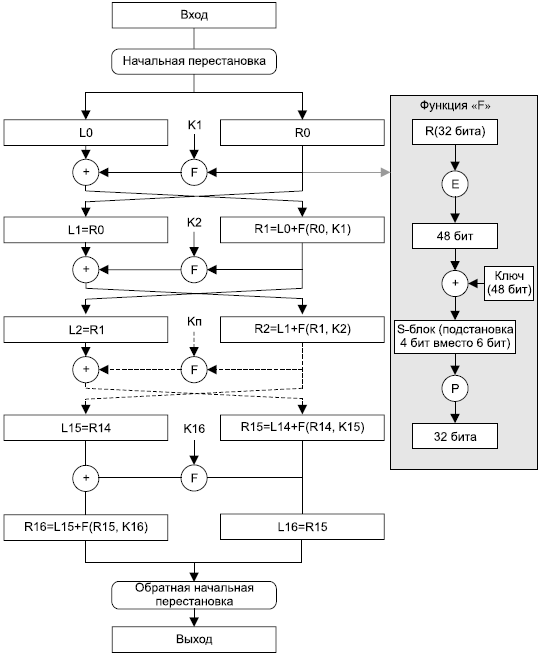
Начнём с трёх ключей:

* DES-EEE3: Исходный текст шифруется три раза, используя разные ключи.
* DES-EDE3: Исходный текст шифруется, затем дешифруется (уже другим ключом), затем снова шифруется (третьим ключом).
* DES-EEE2: Исходный текст шифруется три раза, однако ключи на первом и последнем шаге одинаковые.
* DES-EDE2: Исходный текст шифруется, затем дешифруется (другим ключом), затем снова шифруется (ключом, используемым при первом шифровании).

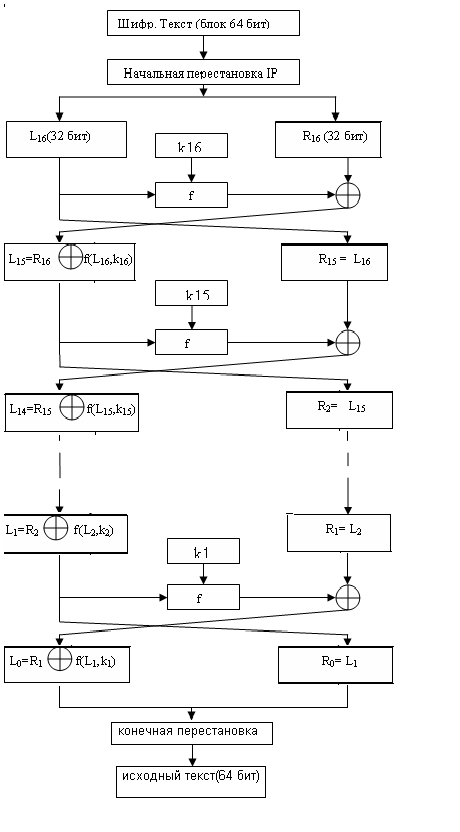
На практике, самый используемый тип 3DES шифрования - DES-EDE3.

**3. Блок-схема алгоритма**

Общая схема алгоритма *кодирования* для DES:



Общая схема алгоритма *декодирования* для DES:



**4. Пример работы программы**

Рассмотрим пример программы.

В исходном файле содержится следующий текст:

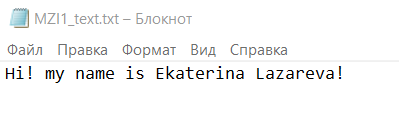


Рисунок 1. Исходный файл



Рисунок 2. Результат шифрования при помощи DES



Рисунок 3. Результат шифрования при помощи Double DES



Рисунок 4. Результат шифрования при помощи Triple DES

**5. Код программы**

import sys

class \_baseDes(object):

def \_\_init\_\_(self, pad=None):

if pad:

pad = self.mustBeBytes(pad)

self.block\_size = 8

self.\_padding = pad

def getKey(self):

return self.\_\_key

def setKey(self, key):

key = self.mustBeBytes(key)

self.\_\_key = key

def getPadding(self):

return self.\_padding

def setPadding(self, pad):

if pad is not None:

pad = self.mustBeBytes(pad)

self.\_padding = pad

def padData(self, data, pad):

if len(data) % self.block\_size == 0:

return data

if not pad:

pad = self.getPadding()

if not pad:

raise ValueError("Data must be a multiple of " + str(self.block\_size) + " bytes in length. Set the pad character.")

data += (self.block\_size - (len(data) % self.block\_size)) \* pad

return data

def unpadData(self, data, pad):

if not data:

return data

if not pad:

pad = self.getPadding()

if pad:

data = data[:-self.block\_size] + \

data[-self.block\_size:].rstrip(pad)

return data

def mustBeBytes(self, data):

if isinstance(data, str):

try:

return data.encode('ascii')

except UnicodeEncodeError:

pass

raise ValueError("pyDes can only work with encoded strings, not Unicode.")

return data

class des(\_baseDes):

Pc1 = [56, 48, 40, 32, 24, 16, 8,

0, 57, 49, 41, 33, 25, 17,

9, 1, 58, 50, 42, 34, 26,

18, 10, 2, 59, 51, 43, 35,

62, 54, 46, 38, 30, 22, 14,

6, 61, 53, 45, 37, 29, 21,

13, 5, 60, 52, 44, 36, 28,

20, 12, 4, 27, 19, 11, 3

]

leftRotations = [

1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1

]

Pc2 = [

13, 16, 10, 23, 0, 4,

2, 27, 14, 5, 20, 9,

22, 18, 11, 3, 25, 7,

15, 6, 26, 19, 12, 1,

40, 51, 30, 36, 46, 54,

29, 39, 50, 44, 32, 47,

43, 48, 38, 55, 33, 52,

45, 41, 49, 35, 28, 31

]

# initial permutation IP

IP = [57, 49, 41, 33, 25, 17, 9, 1,

59, 51, 43, 35, 27, 19, 11, 3,

61, 53, 45, 37, 29, 21, 13, 5,

63, 55, 47, 39, 31, 23, 15, 7,

56, 48, 40, 32, 24, 16, 8, 0,

58, 50, 42, 34, 26, 18, 10, 2,

60, 52, 44, 36, 28, 20, 12, 4,

62, 54, 46, 38, 30, 22, 14, 6

]

# Expansion table for turning 32 bit blocks into 48 bits

E = [

31, 0, 1, 2, 3, 4,

3, 4, 5, 6, 7, 8,

7, 8, 9, 10, 11, 12,

11, 12, 13, 14, 15, 16,

15, 16, 17, 18, 19, 20,

19, 20, 21, 22, 23, 24,

23, 24, 25, 26, 27, 28,

27, 28, 29, 30, 31, 0

]

S\_box = [

# S1

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

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

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

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

# S2

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

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

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

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

# S3

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

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

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

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

# S4

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

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

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

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

# S5

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

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

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

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

# S6

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

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

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

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

# S7

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

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

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

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

# S8

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

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

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

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

]

P = [

15, 6, 19, 20, 28, 11,

27, 16, 0, 14, 22, 25,

4, 17, 30, 9, 1, 7,

23,13, 31, 26, 2, 8,

18, 12, 29, 5, 21, 10,

3, 24

]

IP\_reversed = [

39, 7, 47, 15, 55, 23, 63, 31,

38, 6, 46, 14, 54, 22, 62, 30,

37, 5, 45, 13, 53, 21, 61, 29,

36, 4, 44, 12, 52, 20, 60, 28,

35, 3, 43, 11, 51, 19, 59, 27,

34, 2, 42, 10, 50, 18, 58, 26,

33, 1, 41, 9, 49, 17, 57, 25,

32, 0, 40, 8, 48, 16, 56, 24

]

ENCRYPT = 0

DECRYPT = 1

def \_\_init\_\_(self, key, pad=None):

# Sanity checking of arguments.

if len(key) != 8:

raise ValueError("Invalid DES key size. Key must be exactly 8 bytes long.")

\_baseDes.\_\_init\_\_(self, pad)

self.key\_size = 8

self.L = []

self.R = []

self.Kn = [ [0] \* 48 ] \* 16 # 16 48-bit keys (K1 - K16)

self.final = []

self.setKey(key)

def setKey(self, key):

\_baseDes.setKey(self, key)

self.createSubKeys()

def stringToBits(self, data):

l = len(data) \* 8

result = [0] \* l

pos = 0

for ch in data:

i = 7

while i >= 0:

if ch & (1 << i) != 0:

result[pos] = 1

else:

result[pos] = 0

pos += 1

i -= 1

return result

def bitsToString(self, data):

result = []

pos = 0

c = 0

while pos < len(data):

c += data[pos] << (7 - (pos % 8))

if (pos % 8) == 7:

result.append(c)

c = 0

pos += 1

return bytes(result)

def permute(self, table, block):

return list(map(lambda x: block[x], table))

def createSubKeys(self):

key = self.permute(des.Pc1, self.stringToBits(self.getKey()))

i = 0

# Split into Left and Right sections

self.L = key[:28]

self.R = key[28:56]

while i < 16:

j = 0

# Perform circular left shifts

while j < des.leftRotations[i]:

self.L.append(self.L[0])

del self.L[0]

self.R.append(self.R[0])

del self.R[0]

j += 1

# Create one of the 16 subkeys through pc2 permutation

self.Kn[i] = self.permute(des.Pc2, self.L + self.R)

i += 1

def desCrypt(self, block, crypt\_type):

block = self.permute(des.IP, block)

self.L = block[:32]

self.R = block[32:]

if crypt\_type == des.ENCRYPT:

iteration = 0

iteration\_adjustment = 1

else:

iteration = 15

iteration\_adjustment = -1

i = 0

while i < 16:

tempR = self.R[:]

self.R = self.permute(des.E, self.R)

self.R = list(map(lambda x, y: x ^ y, self.R, self.Kn[iteration]))

B = [self.R[:6], self.R[6:12], self.R[12:18], self.R[18:24], self.R[24:30], self.R[30:36], self.R[36:42], self.R[42:]]

j = 0

Bn = [0] \* 32

pos = 0

while j < 8:

m = (B[j][0] << 1) + B[j][5]

n = (B[j][1] << 3) + (B[j][2] << 2) + (B[j][3] << 1) + B[j][4]

v = des.S\_box[j][(m << 4) + n]

Bn[pos] = (v & 8) >> 3

Bn[pos + 1] = (v & 4) >> 2

Bn[pos + 2] = (v & 2) >> 1

Bn[pos + 3] = v & 1

pos += 4

j += 1

self.R = self.permute(des.P, Bn)

self.R = list(map(lambda x, y: x ^ y, self.R, self.L))

self.L = tempR

i += 1

iteration += iteration\_adjustment

self.final = self.permute(des.IP\_reversed, self.R + self.L)

return self.final

def crypt(self, data, crypt\_type):

if not data:

return ''

if len(data) % self.block\_size != 0:

if crypt\_type == des.DECRYPT: # Decryption must work on 8 byte blocks

raise ValueError("Invalid data length, data must be a multiple of " + str(self.block\_size) + " bytes\n.")

if not self.getPadding():

raise ValueError("Invalid data length, data must be a multiple of " + str(self.block\_size) + " bytes\n. Try setting the optional padding character")

else:

data += (self.block\_size - (len(data) % self.block\_size)) \* self.getPadding()

i = 0

dict = {}

result = []

while i < len(data):

block = self.stringToBits(data[i:i+8])

processed\_block = self.desCrypt(block, crypt\_type)

result.append(self.bitsToString(processed\_block))

i += 8

return bytes.fromhex('').join(result)

def encrypt(self, data, pad=None):

data = self.mustBeBytes(data)

if pad is not None:

pad = self.mustBeBytes(pad)

data = self.padData(data, pad)

return self.crypt(data, des.ENCRYPT)

def decrypt(self, data, pad=None):

data = self.mustBeBytes(data)

if pad is not None:

pad = self.mustBeBytes(pad)

data = self.crypt(data, des.DECRYPT)

return self.unpadData(data, pad)

class doubleDes(\_baseDes):

def \_\_init\_\_(self, key, pad=None):

\_baseDes.\_\_init\_\_(self, pad)

self.key\_size = 16

self.setKey(key)

def setKey(self, key):

if len(key) != self.key\_size:

raise ValueError("Invalid double DES key size. Key must be 16 bytes long")

self.\_\_key1 = des(key[:8], self.\_padding)

self.\_\_key2 = des(key[8:], self.\_padding)

\_baseDes.setKey(self, key)

def setPadding(self, pad):

\_baseDes.setPadding(self, pad)

for key in (self.\_\_key1, self.\_\_key2):

key.setPadding(pad)

def encrypt(self, data, pad=None):

ENCRYPT = des.ENCRYPT

DECRYPT = des.DECRYPT

data = self.mustBeBytes(data)

if pad is not None:

pad = self.mustBeBytes(pad)

# Pad the data accordingly.

data = self.padData(data, pad)

data = self.\_\_key1.crypt(data, ENCRYPT)

return self.\_\_key2.crypt(data, DECRYPT)

def decrypt(self, data, pad=None):

ENCRYPT = des.ENCRYPT

DECRYPT = des.DECRYPT

data = self.mustBeBytes(data)

if pad is not None:

pad = self.mustBeBytes(pad)

data = self.\_\_key2.crypt(data, ENCRYPT)

data = self.\_\_key1.crypt(data, DECRYPT)

return self.unpadData(data, pad)

class tripleDes(\_baseDes):

def \_\_init\_\_(self, key, pad=None):

\_baseDes.\_\_init\_\_(self, pad)

self.key\_size = 24

self.setKey(key)

def setKey(self, key):

if len(key) != self.key\_size:

raise ValueError("Invalid triple DES key size. Key must be 24 bytes long")

self.\_\_key1 = des(key[:8], self.\_padding)

self.\_\_key2 = des(key[8:16], self.\_padding)

self.\_\_key3 = des(key[16:], self.\_padding)

\_baseDes.setKey(self, key)

def setPadding(self, pad):

\_baseDes.setPadding(self, pad)

for key in (self.\_\_key1, self.\_\_key2, self.\_\_key3):

key.setPadding(pad)

def encrypt(self, data, pad=None):

ENCRYPT = des.ENCRYPT

DECRYPT = des.DECRYPT

data = self.mustBeBytes(data)

if pad is not None:

pad = self.mustBeBytes(pad)

# Pad the data accordingly.

data = self.padData(data, pad)

data = self.\_\_key1.crypt(data, ENCRYPT)

data = self.\_\_key2.crypt(data, DECRYPT)

return self.\_\_key3.crypt(data, ENCRYPT)

def decrypt(self, data, pad=None):

ENCRYPT = des.ENCRYPT

DECRYPT = des.DECRYPT

data = self.mustBeBytes(data)

if pad is not None:

pad = self.mustBeBytes(pad)

data = self.\_\_key3.crypt(data, DECRYPT)

data = self.\_\_key2.crypt(data, ENCRYPT)

data = self.\_\_key1.crypt(data, DECRYPT)

return self.unpadData(data, pad)

file = open("MZI1\_text.txt", "r")

data = file.read().encode()

print(data)

k = Des(b"DESCRYPT", pad=b"\1")

d = k.encrypt(data)

print("Encrypted: %r" % d)

print("Decrypted: %r" % k.decrypt(d))

assert k.decrypt(d) == data

k = DoubleDes(b"DESCRYPTREALREAL", pad=b"\1")

d = k.encrypt(data)

print("Encrypted: %r" % d)

print("Decrypted: %r" % k.decrypt(d))

assert k.decrypt(d) == data

k = TripleDes(b"DESCRYPTREALREALHOOLHOOL", pad=b"\1")

d = k.encrypt(data)

print("Encrypted: %r" % d)

print("Decrypted: %r" % k.decrypt(d))

assert k.decrypt(d) == data

**6. Вывод**

DES был национальным стандартом США в 1977—1980 гг., но в настоящее время DES используется (с ключом длины 56 бит) только для устаревших систем, чаще всего используют его более криптоустойчивый вид (3DES, DESX). 3DES является простой эффективной заменой DES, и сейчас он рассмотрен как стандарт. В ближайшее время DES и Triple DES будут заменены алгоритмом AES (Advanced Encryption Standard — Расширенный Стандарт Шифрования).