МИНОБРНАУКИ РОССИИ

Изображение выглядит как текст

Автоматически созданное описаниеФедеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра информатики и систем управления

Отчет

По лабораторной работе №3

«Шифр DES»

По дисциплине

«Защита информации»

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Капранов С.Н.

(подпись) (фамилия, и.,о.)

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Степушин С.В.

(подпись) (фамилия, и.,о.)

19-ВМ

(шифр группы)

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

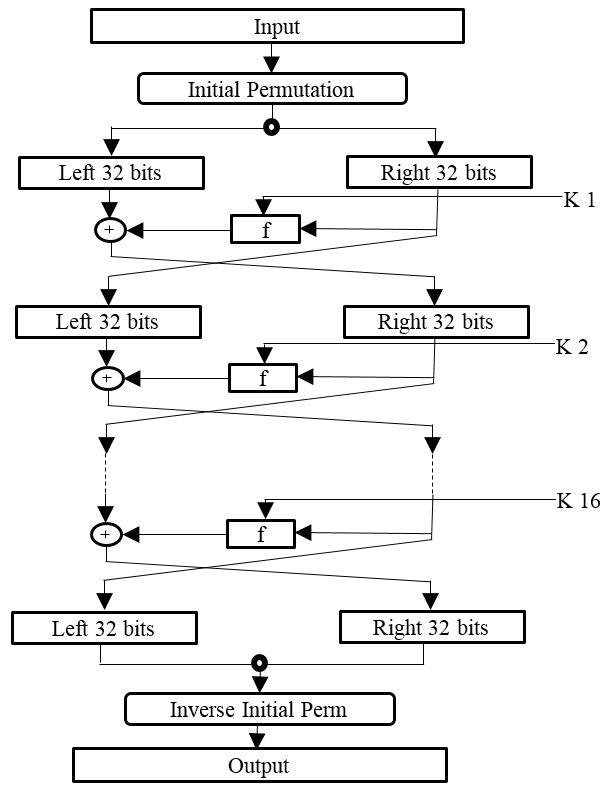
Нижний Новгород 2022

**Задание №3.1.**

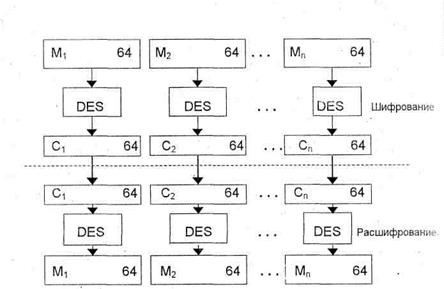
Реализовать стандарт шифрования данных DES в режиме «Электронная кодовая книга».

**DES в режиме «Электронная кодовая книга».**

Алгоритм DES использует комбинацию подстановок и перестановок. DES осуществляет шифрование 64-битовых блоков данных с помощью 64-битового ключа, в котором значащими являются 56 бит (остальные 8 бит — проверочные биты для контроля на четность). Дешифрование в DES является операцией, обратной шифрованию, и выполняется путем повторения операций шифрования в обратной последовательности. Обобщенная схема процесса:



Режим «Электронная кодовая книга» заключается в следующем: длинный файл разбивают на 64-битовые отрезки (блоки) по 8 байт. Каждый из этих блоков шифруют независимо с использованием одного и того же ключа шифрования. Схема алгоритма:



**Алгоритм.**

**Общий алгоритм**:

1. Вводим сообщение, которое будем кодировать/декодировать.

2. Вводим ключ, конвертируем в биты.

3. Вычисляем количество блоков для кодирования.

4. Выбираем первые 8 символов.

5. Конвертируем в биты.

6. Кодируем в битах.

7 Конвертируем в строку.

8. Вычисляем количество блоков для декодирования.

9. Выбираем первые 8 символов.

10. Конвертируем в биты.

11. Декодируем в битах.

12 Конвертируем в строку.

**Кодирование**: получаем на вход последовательность бит сообщения и последовательность бит ключа.

1. Сначала производим первоначальную перестановку в соответствии с таблицей IP.

2. Разделяем последовательность бит на левую (L, 0-31) и правую (R,32-63).

3. Из ключа переставляем биты в две последовательности C и D по 28 бит.

4. Запускаем цикл на 16 итераций:

* Сохраняем последовательность R.
* Генерируем ключ Ki.
* Вычисляем последовательность f(R,Ki).
* Выполняем исключающее ИЛИ между L и f.
* Меняем местами L и R (кроме последней итерации).

5. Выполняем конечную перестановку в соответствии с таблицей IP-1.

**Декодирование**: получаем на вход последовательность бит зашифрованного сообщения и последовательность бит ключа.

1. Сначала производим перестановку обратную конечной перестановке в соответствии с таблицей IP-1.

2. Разделяем последовательность бит на левую (L, 0-31) и правую (R,32-63).

3. Из ключа переставляем биты в две последовательности C и D по 28 бит.

4. Генерируем все ключи Ki и сохраняем в вектор.

5. Производим 0 итерацию вне общего цикла:

* Вытаскиваем последний ключ Ki из вектора.
* Удаляем последний ключ Ki из вектора.
* Вычисляем последовательность f(R,Ki).
* Выполняем исключающее ИЛИ между L и f.

6. Запускаем цикл на оставшиеся 15 итераций:

* Сохраняем последовательность L.
* Вытаскиваем последний ключ Ki из вектора.
* Удаляем последний ключ Ki из вектора.
* Вычисляем последовательность f(L,Ki).
* Выполняем исключающее ИЛИ между R и f.
* Меняем местами L и R.

7. Выполняем перестановку обратную первичной перестановке в соответствии с таблицей IP.

**Функция шифрования f**: на вход получаем последовательность бит половины сообщения и ключ Ki.

1. Сначала производим расширение последовательности с 32 бит до 48 бит в соответствии с таблицей E.

2. Затем выполняем исключающее ИЛИ между получившейся последовательность E и ключом Ki.

3. Производим трансформацию S:

* Производим разбиение получившейся последовательности EK на 8 блоков по 6 битов.
* Каждый блок мы преобразуем из 6 бит в 4 бита в зависимости от таблиц Si.
* Собираем последовательность S из восьми четырех битных блоков Si.

4. Над последовательностью S выполняем перестановку бит P в соответствии с таблицей P.

**Генерация ключей Ki:** на вход поступают две последовательности C и D.

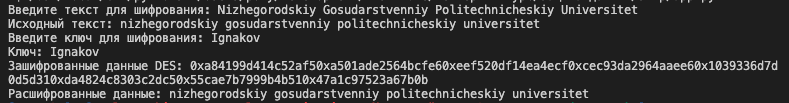
1. Производим сдвиг влево на 1 или 2 бита (в зависимости от итерации) в последовательностях C и D.

2. Затем собираем одну последовательность CD из 56 бит.

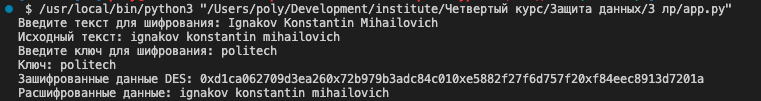
3. И переставляем биты в соответствии с таблицей H. Получается последовательность Ki из 48 бит (8 бит мы не задействуем).

**Примеры шифрования/расшифровывания.**

1. Nizhegorodskiy Gosudarstvenniy Politechnicheskiy Universitet:



2.Ignakov Konstantin Mihailovich:



**Текст программы.**

from bitarray import bitarray

class MyDES:

def \_\_init\_\_(self):

self.child\_keys = []

@staticmethod

def \_bit\_encode(s: str) -> str:

"""

Преобразуйте строку в строковую форму 01

"""

return bitarray(

''.join([bin(int('1' + hex(c)[2:], 16))[3:]

for c in s.encode('utf-8')])).to01()

@staticmethod

def \_bit\_decode(s: list):

return ''.join([chr(i) for i in [int(b, 2) for b in s]])

@staticmethod

def \_negate(s: str):

result = ""

try:

for i in s:

result += '0' if i == '1' else '1'

return result

except:

print("Ошибка ключа")

raise

@staticmethod

def \_replace\_block(block: str, replace\_table: tuple) -> str:

"""

Заменить отдельный блок

Args:

block: str, 64-битная строка 01 для преобразования

replace\_table: таблица преобразования

Return:

Вернуть преобразованную строку

"""

result = ""

for i in replace\_table:

try:

result += block[i - 1]

except IndexError:

print(i)

# print(f"block= {block}, len={len(block)}")

raise

return result

def \_processing\_encode\_input(self, enter: str) -> list:

"""

Преобразуйте входную строку в двоичную форму и разделите ее на группу без 64 ​​бит

"""

result = []

bit\_string = self.\_bit\_encode(enter)

# Если длина не делится на 64, добавить ноль

if len(bit\_string) % 64 != 0:

for i in range(64 - len(bit\_string) % 64):

bit\_string += '0'

for i in range(len(bit\_string) // 64):

result.append(bit\_string[i \* 64: i \* 64 + 64])

# print (f "Исходный открытый текст преобразован в двоичный: {результат}")

return result

@staticmethod

def \_processing\_decode\_input(enter: str) -> list:

result = []

try:

input\_list = enter.split("0x")[1:]

int\_list = [int("0x" + i, 16) for i in input\_list]

for i in int\_list:

bin\_data = str(bin(i))[2:]

while len(bin\_data) < 64:

bin\_data = '0' + bin\_data

result.append(bin\_data)

return result

except Exception as e:

raise

def \_key\_conversion(self, key: str):

"""

Преобразуйте 64-битный исходный ключ в 56-битный ключ и выполните замену

"""

key = self.\_bit\_encode(key)

while len(key) < 64:

key += '0'

first\_key = key[:64]

key\_replace\_table = (

57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34, 26, 18,

10, 2, 59, 51, 43, 35, 27, 19, 11, 3, 60, 52, 44, 36,

63, 55, 47, 39, 31, 23, 15, 7, 62, 54, 46, 38, 30, 22,

14, 6, 61, 53, 45, 37, 29, 21, 13, 5, 28, 20, 12, 4

)

return self.\_replace\_block(first\_key, key\_replace\_table)

def \_spin\_key(self, key: str):

"""

Поверните, чтобы получить подключ

"""

kc = self.\_key\_conversion(key)

first, second = kc[0: 28], kc[28: 56]

spin\_table = (1, 2, 4, 6, 8, 10, 12, 14, 15, 17, 19, 21, 23, 25, 27, 28)

for i in range(1, 17):

first\_after\_spin = first[spin\_table[i - 1]:] + first[:spin\_table[i - 1]]

second\_after\_spin = second[spin\_table[i - 1]:] + second[:spin\_table[i - 1]]

# print (f "Ключ после поворота: слева: {first\_after\_spin}, справа: {second\_after\_spin}")

yield first\_after\_spin + second\_after\_spin

def \_key\_selection\_replacement(self, key: str):

"""

Получите 48-битный подключ, выбрав перестановку

"""

# Сначала оставьте пустым

self.child\_keys = []

key\_select\_table = (

14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21, 10,

23, 19, 12, 4, 26, 8, 16, 7, 27, 20, 13, 2,

41, 52, 31, 37, 47, 55, 30, 40, 51, 45, 33, 48,

44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32

)

for child\_key56 in self.\_spin\_key(key):

self.child\_keys.append(self.\_replace\_block(child\_key56, key\_select\_table))

def \_init\_replace\_block(self, block: str):

"""

Выполнить замену начального состояния на блоке

"""

replace\_table = (

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

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

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

64, 56, 48, 40, 32, 24, 16, 8,

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

)

return self.\_replace\_block(block, replace\_table)

def \_end\_replace\_block(self, block: str) -> str:

"""

Конвертация конечного состояния блока

"""

replace\_table = (

40, 8, 48, 16, 56, 24, 64, 32,

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

)

return self.\_replace\_block(block, replace\_table)

@staticmethod

def \_block\_extend(block: str) -> str:

"""

Расширенная замена

"""

extended\_block = ""

extend\_table = (

32, 1, 2, 3, 4, 5,

4, 5, 6, 7, 8, 9,

8, 9, 10, 11, 12, 13,

12, 13, 14, 15, 16, 17,

16, 17, 18, 19, 20, 21,

20, 21, 22, 23, 24, 25,

24, 25, 26, 27, 28, 29,

28, 29, 30, 31, 32, 1

)

for i in extend\_table:

extended\_block += block[i - 1]

return extended\_block

@staticmethod

def \_not\_or(a: str, b: str) -> str:

"""

XOR двух строк 01

"""

result = ""

size = len(a) if len(a) < len(a) else len(b)

for i in range(size):

result += '0' if a[i] == b[i] else '1'

return result

def \_s\_box\_replace(self, block48: str) -> str:

"""

Замена блока S, преобразование 48-битного ввода в 32-битный вывод

"""

s\_box\_table = (

(

(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),

),

(

(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),

),

(

(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),

),

(

(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),

),

(

(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),

),

(

(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),

),

(

(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),

),

(

(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),

)

)

result = ""

for i in range(8):

row\_bit = (block48[i \* 6] + block48[i \* 6 + 5]).encode("utf-8")

line\_bit = (block48[i \* 6 + 1: i \* 6 + 5]).encode("utf-8")

row = int(row\_bit, 2)

line = int(line\_bit, 2)

# print (f "Строка {строка}, столбец {строка}")

data = s\_box\_table[i][row][line]

no\_full = str(bin(data))[2:]

while len(no\_full) < 4:

no\_full = '0' + no\_full

result += no\_full

return result

def \_s\_box\_compression(self, num: int, block48: str) -> str:

"""

После расширения и замены сжатие S-блока выполняется на 48-битной строке 01. Состоит из двух частей:

1. XOR с ключом

2. Согласно таблице сжатия S-блока, 48 бит сжимаются до 36 бит.

Args:

число: первая итерация

block48: right

Return:

Вернуть 32-битную строку 01, сжатую блоком S

"""

result\_not\_or = self.\_not\_or(block48, self.child\_keys[num])

# print (f "Результат XOR с ключом {result\_not\_or}")

return self.\_s\_box\_replace(result\_not\_or)

def \_p\_box\_replacement(self, block32: str) -> str:

"""

Замена коробки P

Return:

Вернуть 32-битную строку 01 после замены блока P

"""

p\_box\_replace\_table = (

16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23, 26, 5, 18, 31, 10,

2, 8, 24, 14, 32, 27, 3, 9, 19, 13, 30, 6, 22, 11, 4, 25,

)

return self.\_replace\_block(block32, p\_box\_replace\_table)

def \_f\_function(self, right: str, is\_decode: bool, num: int):

right = self.\_block\_extend(right)

if is\_decode:

sbc\_result = self.\_s\_box\_compression(15 - num, right)

else:

sbc\_result = self.\_s\_box\_compression(num, right)

# print (f "Сжатый результат блока s: {sbc\_result}")

return self.\_p\_box\_replacement(sbc\_result)

def \_iteration(self, block: str, key: str, is\_decode: bool) -> str:

self.\_key\_selection\_replacement(key)

for i in range(16):

left, right = block[0: 32], block[32: 64]

next\_left = right

f\_result = self.\_f\_function(right, is\_decode, i)

right = self.\_not\_or(left, f\_result)

block = next\_left + right

return block[32:] + block[:32]

def encode(self, enter: str, key: str):

result = ""

blocks = self.\_processing\_encode\_input(enter)

for block in blocks:

irb\_result = self.\_init\_replace\_block(block)

block\_result = self.\_iteration(irb\_result, key, is\_decode=False)

block\_result = self.\_end\_replace\_block(block\_result)

result += str(hex(int(block\_result.encode(), 2)))

return result

def decode(self, cipher\_text: str, key: str):

result = []

blocks = self.\_processing\_decode\_input(cipher\_text)

for block in blocks:

irb\_result = self.\_init\_replace\_block(block)

block\_result = self.\_iteration(irb\_result, key, is\_decode=True)

block\_result = self.\_end\_replace\_block(block\_result)

for i in range(0, len(block\_result), 8):

result.append(block\_result[i: i+8])

return self.\_bit\_decode(result)

if \_\_name\_\_ == '\_\_main\_\_':

print('Исходный текст:', text := input("Введите текст для шифрования: ").strip().lower().replace('ё', 'е'))

print('Ключ:', key := str(input("Введите ключ для шифрования: ")))

md = MyDES()

print("Зашифрованные данные DES: " + (code := md.encode(text, key)))

print(f"Расшифрованные данные: {md.decode(code, key)}")