НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра прикладної математики

Звіт

з лабораторної роботи № 3

із дисципліни «Криптографічні методи захисту інформації»

на тему

*Стандарт шифрування DES*

|  |  |
| --- | --- |
| Виконала: | Керівник: |
| студентка групи КМ-81 | *ст. викладач Бай Ю. П.* |
| *Верзун П.В.* |  |

Київ — 20\_\_

ЗМІСТ

[Постановка задачі 2](#_Toc53614193)

[Основні теоретичні відомості зі стандарту шифрування DES 2](#_Toc53614194)

[Контрольний приклад 1 3](#_Toc53614195)

[Контрольний приклад 2 4](#_Toc53614196)

[Шифрування тексту 5](#_Toc53614197)

[Розшифрування тексту 5](#_Toc53614198)

[Список літератури 6](#_Toc53614199)

[Додаток 1 7](#_Toc53614200)

[Додаток 2 8](#_Toc53614201)

***Мета роботи:*** розробити криптосистему на основі стандарту шифрування DES.

# Постановка задачі

1. Скласти програму для шифрування та розшифрування за алгоритмом DES 64-бітного блоку інформації, використовуючи 64-бітний ключ. Перевірити роботу програми на контрольних прикладах.

1.*а*. Контрольний приклад 1

*plaintext* = 01 23 45 67 89 AB CD EF (hex)

*key* = 13 34 57 79 9B BC DF F1 (hex)

*ciphertext* = 85 E8 13 54 0F 0A B4 05 (hex)

1.*б*. Контрольний приклад 2

*plaintext* = 01 23 45 67 89 AB CD EF (hex)

*key* = FE DC BA 98 76 54 32 10 (hex)

*ciphertext* = ED 39 D9 50 FA 74 BC C4 (hex)

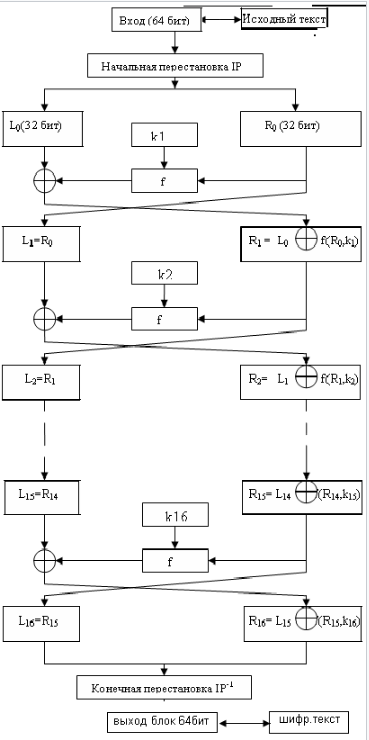
2. Розширити функціональність програми для випадку відкритого тексту довільної довжини та 64-бітного ключа.

3. За стандартом DES зашифрувати текст довжиною від 20 до 80 символів, користуючись 64-бітним ключем. Навести ключ та зашифроване повідомлення в [Таблиці](https://docs.google.com/spreadsheets/d/1Xc_7zofsSJbqkxv4w_WagGuF0VnuFISEO-SqDzCAXLI/edit?usp=sharing) в своєму рядку. **Зауваження:** *key* та *ciphertext* надавати у вигляді repr() або hex.

4. Дано зашифроване за стандартом DES повідомлення довжиною від 20 до 80 символів та відомий 64-бітний ключ (див. [Таблицю](https://docs.google.com/spreadsheets/d/1Xc_7zofsSJbqkxv4w_WagGuF0VnuFISEO-SqDzCAXLI/edit?usp=sharing), рядок над своїм). Розшифрувати задане повідомлення. Результат записати в [Таблицю](https://docs.google.com/spreadsheets/d/1Xc_7zofsSJbqkxv4w_WagGuF0VnuFISEO-SqDzCAXLI/edit?usp=sharing), в рядок над своїм.

# Основні теоретичні відомості зі стандарту шифрування DES

DES є блочним шифром - дані шифруються блоками по 64 біти - 64 бітний блок явного тексту подається на вхід алгоритму, а 64-бітний блок шифрограми отримується в результаті роботи алгоритму. Крім того, як під час шифрування, так і під час дешифрування використовується один і той самий алгоритм (за винятком дещо іншого шляху утворення робочих ключів).

Ключ має довжину 56 біт (як правило, в джерельному вигляді ключ має довжину 64 біти, де кожний 8-й біт є бітом паритету, крім того, ці контрольні біти можуть бути винесені в останній байт ключа). Ключем може бути довільна 64-бітна комбінація, яка може бути змінена у будь-який момент часу. Частина цих комбінацій вважається слабкими ключами, оскільки може бути легко визначена. Безпечність алгоритму базується на безпечності ключа.

На найнижчому рівні алгоритм є ніщо інше, ніж поєднання двох базових технік шифрування: перемішування і підстановки. Цикл алгоритму, з яких і складається DES є комбінацією цих технік, коли як об'єкти перемішування виступають біти тексту, ключа і блоків підстановок.

На рисунку справа приведена схема шифрування алгоритму DES.

# Перевірка реалізації на контрольних прикладах.

Перший контрольний приклад виглядає наступним чином:

* *plaintext* = 01 23 45 67 89 AB CD EF (hex);
* *key* = 13 34 57 79 9B BC DF F1 (hex);
* *ciphertext* = 85 E8 13 54 0F 0A B4 05 (hex).

Другий контрольний приклад виглядає так:

* *plaintext* = 01 23 45 67 89 AB CD EF (hex)
* *key* = FE DC BA 98 76 54 32 10 (hex)
* *ciphertext* = ED 39 D9 50 FA 74 BC C4 (hex)

Для даної реалізації методу необхідно перетворити вхідні дані з класу str в клас bytes. Для цього було використано функцію bytes.fromhex() для вхідного тексту та ключа. Метод описаний таким чином, що все працює відносно певного ключа, тобто ініціалізується об’єкт класу DesKey з одним вхідним параметром – ключем. Далі необхідно викликати метод encrypt цього об’єкту з з єдиним параметром – plaintext. Результат перетворюєтсья в hex та передатсья у фукцію hex\_view, я ка приймає hex-рядок та розбиває його на пари для більшої читабельності.

Таким чином обробляються дані двох контрольних прикладів. Результат виконання програми для кожного з прикладів продемонстровано на рис. 1.

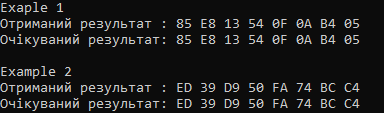


Рисунок 1

# Робота з текстом довільної довжини

В цьому розділі буде продемонстровано рішення проблеми необхідності методу encrypt приймати лише тексти довжини кратної 8. Для цього при виклику функції шифрування необхідно вказати параметр padding=True. В цьому випадку автоматично додаватимуть символи-заповнювачі, а саме hex представлення числа, якого не вистачає до кратності 8, в кінець повідомлення, що шифрується.

В прикладі, що приведено на рисунку 2, показано роботу методів розшифровки при padding=False (рядки 1-2) та padding=True (рядки 4-5). В другому випадку метод «з’їдає» тимчасові дані, і повертає лише декодовані повідомлення. Потрібно відмітити, що padding=True в методі шифрування, бо інакше методи просто не спрацювали б.

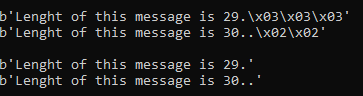


Рисунок 2

Очевидно, що до довжини кратності 8, в першому повідомленні, не вистачає 3 символи, а в другому – 2.

# Шифрування повідомлення

Наступним кроком необхідно придумати та зашифрувати повідомлення для інших студентів. Для цього були обрані наступні ключ та повідомлення:

* key = 'sikorsky'
* message = 'Don\'t look at her. Breasts are temporary. KPI is forever. Live. Love. KPI. (c)Me'

Результат DES-шифрування приведений нижче:

7D E6 11 F9 86 2C A1 D5 9B 9A 99 FB 2D AA FA 1D 46 BC 71 16 43 D7 73 40 05 F4 E1 FF 57 8C 11 D5 AC 23 00 B5 98 96 83 28 72 A8 1B 82 B8 31 D6 0F E5 71 97 1C C4 94 CB 28 A5 A2 D7 89 54 10 DC C8 2E A2 B7 70 9F 60 72 E5 64 84 42 FE 99 8A F2 44

# Розшифровування повідомлення

Далі потрібно перейти до таблиці та знайти дані для розшифрування. Для даного варіанту роботи це дані Горбача Костянтина Ігоровича, а саме

* Key='anything'
* Ciphertext = 'FE 47 76 45 FE E0 2E 2E 5A 6D 26 3D 3E 8E CA BA B3 01 58 60 39 92 99 A3 84 DB CB 96 A1 AB 5A D9'

Тут було заховане наступне:

'Life is a series of choices. '

# Висновок

В даній роботі був розглянути метод симетричного шифрування DES. Було продемонстровано його дану реалізацію на контрольних прикладах, які підтвердили правильність реалізації. Було розшифровано повідомлення одного з одногрупників. Т

# Список літератури

1. Тарнавський Ю.А. Технології захисту інформації [Електронний ресурс] / Ю. А. Тарнавський. – Київ: КПІ ім. Ігоря Сікорського, 2018. – 162 с.
2. Шнайер Б. Прикладная криптография: Протоколы, алгоритмы, исходные тексты на языке Си / Б. Шнайер. – М.: Диалектика, 2003. – 610 с.
3. Алферов А.П., Зубов А.Ю., Кузьмин А.С., Черемушкин А.В. Основы криптографии. – М.: Гелиос АРВ, 2001. – 480 с.
4. Столлингс В. Криптография и защита сетей: принципы и практика, 2-е изд.: Пер. с англ. – М.: «Вильямс», 2001. – 672 с.

# Додаток 1

import struct

from .compatibility import number\_type, iter\_range

from .core import derive\_keys, encode\_block

class DesKey(object):

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

self.\_\_encryption\_key = guard\_key(key)

self.\_\_decryption\_key = self.\_\_encryption\_key[::-1]

self.\_\_key = key

def encrypt(self, message, initial=None, padding=False):

return handle(message, self.\_\_encryption\_key, initial, padding, 1)

def decrypt(self, message, initial=None, padding=False):

return handle(message, self.\_\_decryption\_key, initial, padding, 0)

def is\_single(self):

return len(self.\_\_encryption\_key) == 1

def is\_triple(self):

return len(self.\_\_encryption\_key) == 3

def \_\_hash\_\_(self):

return hash((self.\_\_class\_\_, self.\_\_encryption\_key))

def encode(block, key, encryption):

for k in key:

block = encode\_block(block, k, encryption)

encryption = not encryption

return block

def guard\_key(key):

if isinstance(key, bytearray):

key = bytes(key)

assert isinstance(key, bytes), "The key should be `bytes` or `bytearray`"

assert len(key) in (8, 16, 24), "The key should be of length 8, 16, or 24"

k0, k1, k2 = key[:8], key[8:16], key[16:]

if k1 == k2:

return tuple(derive\_keys(k0)),

k2 = k2 or k0

if k1 == k0:

return tuple(derive\_keys(k2)),

return tuple(tuple(derive\_keys(k)) for k in (k0, k1, k2))

def guard\_message(message, padding, encryption):

assert isinstance(message, bytes), "The message should be bytes"

length = len(message)

if encryption and padding:

return message.ljust(length + 8 >> 3 << 3, bytes((8 - (length & 7), )))

assert length & 7 == 0, (

"The length of the message should be divisible by 8"

"(or set `padding` to `True` in encryption mode)"

)

return message

def guard\_initial(initial):

if initial is not None:

if isinstance(initial, bytearray):

initial = bytes(initial)

if isinstance(initial, bytes):

assert len(initial) & 7 == 0, "The initial value should be of length 8(as `bytes` or `bytearray`)"

return struct.unpack(">Q", initial)[0]

assert isinstance(initial, number\_type), "The initial value should be an integer or bytes object"

assert -1 < initial < 1 << 32, "The initial value should be in range [0, 2\*\*32) (as an integer)"

return initial

def handle(message, key, initial, padding, encryption):

message = guard\_message(message, padding, encryption)

initial = guard\_initial(initial)

blocks = (struct.unpack(">Q", message[i: i + 8])[0] for i in iter\_range(0, len(message), 8))

if initial is None:

# ECB

encoded\_blocks = ecb(blocks, key, encryption)

else:

# CBC

encoded\_blocks = cbc(blocks, key, initial, encryption)

ret = b"".join(struct.pack(">Q", block) for block in encoded\_blocks)

return ret[:-ord(ret[-1:])] if not encryption and padding else ret

def ecb(blocks, key, encryption):

for block in blocks:

yield encode(block, key, encryption)

def cbc(blocks, key, initial, encryption):

if encryption:

for block in blocks:

initial = encode(block ^ initial, key, encryption)

yield initial

else:

for block in blocks:

initial, block = block, initial ^ encode(block, key, encryption)

yield block

try:

number\_type = int, long

except NameError:

number\_type = int

try:

iter\_range = xrange

except NameError:

iter\_range = range

import struct

INITIAL\_PERMUTATION = (

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,

)

INVERSE\_PERMUTATION = (

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,

)

EXPANSION = (

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,

)

PERMUTATION = (

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,

)

PERMUTED\_CHOICE1 = (

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,

)

PERMUTED\_CHOICE2 = (

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,

)

SUBSTITUTION\_BOX = (

(

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,

),

)

ROTATES = (

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

)

def rotate\_left(i28, k):

return i28 << k & 0x0fffffff | i28 >> 28 - k

def permute(data, bits, mapper):

ret = 0

for i, v in enumerate(mapper):

if data & 1 << bits - 1 - v:

ret |= 1 << len(mapper) - 1 - i

return ret

def f(block, key):

block = permute(block, 32, EXPANSION) ^ key

ret = 0

for i, box in enumerate(SUBSTITUTION\_BOX):

i6 = block >> 42 - i \* 6 & 0x3f

ret = ret << 4 | box[i6 & 0x20 | (i6 & 0x01) << 4 | (i6 & 0x1e) >> 1]

return permute(ret, 32, PERMUTATION)

def derive\_keys(key):

key, = struct.unpack(">Q", key)

next\_key = permute(key, 64, PERMUTED\_CHOICE1)

next\_key = next\_key >> 28, next\_key & 0x0fffffff

for bits in ROTATES:

next\_key = rotate\_left(next\_key[0], bits), rotate\_left(next\_key[1], bits)

yield permute(next\_key[0] << 28 | next\_key[1], 56, PERMUTED\_CHOICE2)

def encode\_block(block, derived\_keys, encryption):

block = permute(block, 64, INITIAL\_PERMUTATION)

block = block >> 32, block & 0xffffffff

if not encryption:

derived\_keys = reversed(derived\_keys)

for key in derived\_keys:

block = block[1], block[0] ^ f(block[1], key)

return permute(block[1] << 32 | block[0], 64, INVERSE\_PERMUTATION)

from .base import DesKey

def hex\_view(h):

h = h.zfill(len(h) + len(h) % 2)

ans = ' '.join(h[i: i+2] for i in range(0, len(h), 2)).upper()

return ans

def task1():

print('Exaple 1')

plaintext = bytes.fromhex('01 23 45 67 89 AB CD EF')

k = bytes.fromhex('13 34 57 79 9B BC DF F1')

key = DesKey(k)

enc = key.encrypt(plaintext).hex()

print('Отриманий результат :', hex\_view(enc))

print('Очікуваний результат:', '85 E8 13 54 0F 0A B4 05','\n')

print('Example 2')

plaintext = bytes.fromhex('01 23 45 67 89 AB CD EF')

k = bytes.fromhex('FE DC BA 98 76 54 32 10')

key = DesKey(k)

enc = key.encrypt(plaintext).hex()

print('Отриманий результат :', hex\_view(enc))

print('Очікуваний результат:', 'ED 39 D9 50 FA 74 BC C4', '\n\n')

def task2():

key = '12345678'

key = DesKey(key.encode())

message1 = 'Lenght of this message is 29.'

enc = key.encrypt(message1.encode(), padding=True)

print(key.decrypt(enc))

message2 = 'Lenght of this message is 30..'

enc1 = key.encrypt(message2.encode(), padding=True)

print(key.decrypt(enc1), '\n')

print(key.decrypt(enc, padding=True))

print(key.decrypt(enc1, padding=True), '\n')

enc\_t = key.encrypt(message1.encode(), padding=True)

print(enc\_t)

def task3():

\_key = 'sikorsky'

message = 'Don\'t look at her. Breasts are temporary. KPI is forever. Live. Love. KPI. (c)Me'

key = DesKey(\_key.encode())

enc = key.encrypt(message.encode()).hex()

ciphertext = hex\_view(enc)

print('Message=', repr(message))

print('Key=',repr(\_key))

print('Ciphertext=',repr(ciphertext), '\n\n')

def task4():

key = DesKey(b'anything')

ciphertext= bytes.fromhex('FE 47 76 45 FE E0 2E 2E 5A 6D 26 3D 3E 8E CA BA B3 01 58 60 39 92 99 A3 84 DB CB 96 A1 AB 5A D9')

print('Key is \'anything\'')

print('Ciphertext is \'FE 47 76 45 FE E0 2E 2E 5A 6D 26 3D 3E 8E CA BA B3 01 58 60 39 92 99 A3 84 DB CB 96 A1 AB 5A D9\'')

print('Decrypted message is ', repr(key.decrypt(ciphertext).decode()))

\_\_all\_\_ = ['DesKey']

from files.base import DesKey

from files.funcs import \*

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

task1()

task2()

task3()

task4()

# Додаток 2

