# Nama Kelompok:

1. Ian Felix Jonathan S (05311940000008)
2. Dian Arofathi (05311940000011)
3. Shavica Laksana (0531194000013)
4. Daniel Evan (05311940000016)
5. Bryan Yehuda (05311940000025)

# Enkripsi & Deskripsi teks dengan DES pada client-server

Pada sisi client, client akan mengirimkan teks berupa string ke server. Teks yang akan dikirimkan user akan dilakukan enkripsi terlebih dahulu dengan menggunakan metode DES.

Berikut adalah fungsi DES yang kami gunakann

def xor(a: str, b: str) -> str:

return bin(int(a,2) ^ int(b,2))[2:].rjust(len(a), '0')

def split(msg: str, n: int) -> str:

return ' '.join(msg[i:i+n] for i in range(0, len(msg), n))

def shuffle(key: str, table: tuple) -> str:

return "".join(key[i-1] for i in table)

def hex\_to\_bin(h: str) -> str:

return "".join(map(lambda x: bin(int(x, 16))[2:].rjust(4, '0'), list(h)))

def bin\_to\_hex(b: str) -> str:

return "".join(map(lambda x: hex(int(x, 2))[2:], split(b, 4).split()))

fungsi diatas adalah fungsi yang digunakan untuk melakukan convert, memisahkan string dan melakukan shuffle. Lalu kami membuat kode untuk struktur jaringan feistel nya

class Feistel:

def \_\_init\_\_(self, left: str, right: str, keys: list, f, debug=True):

self.L = [left]

self.R = [right]

self.new\_r = []

self.kr = []

self.boxes = []

self.keys = keys

self.f = f

self.p\_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

)

self.e\_bit\_selection\_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

)

self.SBox = (

(

(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)

)

)

self.debug = debug

def round16(self) -> str:

for i in range(16):

self.L.append(self.R[i])

self.new\_r.append(shuffle(self.R[i], self.e\_bit\_selection\_table))

self.kr.append(self.f(self.new\_r[i], self.keys[i+1]))

self.boxes.append(split(self.kr[i], 6).split())

for idx, b in enumerate(self.boxes[-1]):

row = int(b[0]+b[5], 2)

col = int(b[1:5], 2)

self.boxes[-1][idx] = bin(self.SBox[idx][row][col])[2:].rjust(4, '0')

self.R.append(xor(self.L[i], shuffle(''.join(self.boxes[i]), self.p\_table)))

if self.debug:

print("="\*96+"\n")

print("Feistel\n")

for i in range(17):

print(f"L{i}\t\t: {split(self.L[i], 4)}")

print(f"R{i}\t\t: {split(self.R[i], 4)}")

print()

if i < 16:

print(f"#Round {i+1}")

print(f"E(R{i})\t\t: {split(self.new\_r[i], 6)}")

print(f"K{i+1}\t\t: {split(self.keys[i+1], 6)}")

print(f"K{i+1}+E(R{i})\t: {split(self.kr[i], 6)}")

print(f"S-Box\t\t: {' '.join(self.boxes[i])}")

print(f"S-Box-P\t\t: {split(shuffle(''.join(self.boxes[i]), self.p\_table), 4)}")

return self.R[16] + self.L[16]

Setelah itu kami membuat struktur untuk algorimat DES nya

class DES:

def \_\_init\_\_(self, key: str, debug=False):

self.key = hex\_to\_bin(key)

self.pc1 = (

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,

)

self.pc2 = (

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,

)

self.ip\_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,

)

self.shift\_table = (1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1)

self.reverse\_ip\_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

)

self.debug = debug

self.K = [shuffle(self.key, self.pc1)]

self.C = [self.K[0][:28]]

self.D = [self.K[0][28:]]

self.generate\_subkeys()

def encrypt(self, msg: str) -> str:

msg = hex\_to\_bin(msg)

# print(f"msg: {split(msg, 8)}\nkey: {split(self.key, 8)}")

msg = shuffle(msg, self.ip\_table)

if self.debug:

print(f"shuffled msg: {split(msg, 8)}\n")

print("="\*96+"\n")

print(f"K0\t: {split(self.K[0], 7)}")

print(f"C0\t: {split(self.C[0], 7)}")

print(f"D0\t: {split(self.D[0], 7)}")

print()

for i in range(1, 17):

print(f"C{i}\t: {split(self.C[i], 7)}")

print(f"D{i}\t: {split(self.D[i], 7)}")

print(f"C{i}D{i}\t: {split(self.C[i]+self.D[i], 7)}")

print(f"K{i}\t: {split(self.K[i], 6)}")

print()

left = msg[:32]

right = msg[32:]

feistel = Feistel(left, right, self.K, xor, self.debug)

res = feistel.round16()

if self.debug:

print(f"R16L16\t\t: {split(res, 8)}")

res = shuffle(res, self.reverse\_ip\_table)

if self.debug:

print(f"Reverse IP\t: {split(res, 8)}")

print()

print("="\*96)

print()

return bin\_to\_hex(res)

def decrypt(self, ciphertext) -> str:

ciphertext = hex\_to\_bin(ciphertext)

ciphertext = shuffle(ciphertext, self.ip\_table)

left = ciphertext[:32]

right = ciphertext[32:]

feistel = Feistel(left, right, [self.K[0]]+self.K[1:][::-1], xor, self.debug)

res = feistel.round16()

res = shuffle(res, self.reverse\_ip\_table)

return bin\_to\_hex(res)

def generate\_subkeys(self) -> None:

for i in range(16):

shift = self.shift\_table[i]

self.C.append(self.C[i][shift:]+self.C[i][:shift])

self.D.append(self.D[i][shift:]+self.D[i][:shift])

for i in range(16):

self.K.append(shuffle(self.C[i+1] + self.D[i+1], self.pc2))

Dikarenakan untuk memproses DES dibutuhkan input berupa HEX dan yang diinputkan user itu berupa string maka kami membuat sebuah fungsi untuk merubah string menjadi format hex

def string2hex(s):

List=[]

for ch in s:

st2hx = hex(ord(ch)).replace('0x','')

if(len(st2hx)==1): st2hx = '0' + st2hx

List.append(st2hx)

return reduce(lambda i, j: i+j, List)

karena kami menggunakan kunci sepanjang 8 digit, maka kami harus memastikan bahwa inputan user itu panjangnya 8 digit, maka kami membuat sebuah fungsi pad

def pad(text):

while len(text) %8 != 0:

text += ' '

return text

Untuk proses client-server nya kami menggunakan socket programming. Hal yang kami masukkan didalam kode pada client.py

def send(msg):

message = msg.encode(FORMAT)

msg\_length = len(message)

send\_length = str(msg\_length).encode(FORMAT)

send\_length += b' ' \* (HEADER - len(send\_length))

client.send(send\_length)

client.send(message)

print(client.recv(2048).decode(FORMAT))

x = ' '

while x != 'quit':

# Ask the user for a name.

key = "myChiper"

x = ' '

x = input()

pad\_text = pad(x)

msg = string2hex(pad\_text)

key = string2hex(key)

des = DES(key, False)

enc = des.encrypt(msg).upper()

print(f"encrypted: {enc}")

send(enc)

Fungsi send akan digunakan untuk melakukan pengirim teks ke server. Pada fungsi while dapat kita Lihat bahwa, kata kata yang dituliskan oleh user akan dilakukan enkripsi terlebih dahulu menggunakan kode DES yang sebelumnya. Dan hasil enkripsinya tersebut kita kirimkan ke server menggunakan fungsi send.

Untuk server tetap menggunakan socket programming. Untuk mengolah data pada server kami menggunakan kode berikut

def handle\_client(conn, addr):

print(f"[NEW CONNECTION] {addr} connected.")

connected = True

while connected:

msg\_length = conn.recv(HEADER).decode(FORMAT)

if msg\_length:

msg\_length = int(msg\_length)

msg = conn.recv(msg\_length).decode(FORMAT)

if msg == DISCONNECT\_MESSAGE:

connected = False

hex\_string = des.decrypt(msg).upper()

bytes\_object = bytes.fromhex(hex\_string)

ascii\_string = bytes\_object.decode("ASCII")

print(f"[{addr}] {ascii\_string}")

conn.send("Msg received\n".encode(FORMAT))

conn.close()

def start():

server.listen()

print(f"[LISTENING] Server is listening on {SERVER}")

while True:

conn, addr = server.accept()

thread = threading.Thread(target=handle\_client, args=(conn, addr))

thread.start()

print(f"[ACTIVE CONNECTIONS] {threading.activeCount() - 1}")

print("[STARTING] server is starting...")

start()

fungsi start digunakan untuk menginisiasi supaya proses pengiriman data antara client dan server bisa dilaksanakan. Fungsi handle client ini kami gunakan untuk menerima data yang dikirim oleh user. Karena data yang dikirim oleh user masdih berformat string, maka kami convert dulu datanya kedalam bentuk hex lalu dilakukan decode terhadap teks tersebut. Setelah itu, hasil dari decode nya akan ditampilkan pada server

Berikut adalah kode lengkap untuk

1. Server.py

import socket

import threading

from functools import reduce

def string2hex(s):

List=[]

for ch in s:

st2hx = hex(ord(ch)).replace('0x','')

if(len(st2hx)==1): st2hx = '0' + st2hx

List.append(st2hx)

return reduce(lambda i, j: i+j, List)

def xor(a: str, b: str) -> str:

return bin(int(a,2) ^ int(b,2))[2:].rjust(len(a), '0')

def split(msg: str, n: int) -> str:

return ' '.join(msg[i:i+n] for i in range(0, len(msg), n))

def shuffle(key: str, table: tuple) -> str:

return "".join(key[i-1] for i in table)

def hex\_to\_bin(h: str) -> str:

return "".join(map(lambda x: bin(int(x, 16))[2:].rjust(4, '0'), list(h)))

def bin\_to\_hex(b: str) -> str:

return "".join(map(lambda x: hex(int(x, 2))[2:], split(b, 4).split()))

class Feistel:

def \_\_init\_\_(self, left: str, right: str, keys: list, f, debug=True):

self.L = [left]

self.R = [right]

self.new\_r = []

self.kr = []

self.boxes = []

self.keys = keys

self.f = f

self.p\_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

)

self.e\_bit\_selection\_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

)

self.SBox = (

(

(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)

)

)

self.debug = debug

def round16(self) -> str:

for i in range(16):

self.L.append(self.R[i])

self.new\_r.append(shuffle(self.R[i], self.e\_bit\_selection\_table))

self.kr.append(self.f(self.new\_r[i], self.keys[i+1]))

self.boxes.append(split(self.kr[i], 6).split())

for idx, b in enumerate(self.boxes[-1]):

row = int(b[0]+b[5], 2)

col = int(b[1:5], 2)

self.boxes[-1][idx] = bin(self.SBox[idx][row][col])[2:].rjust(4, '0')

self.R.append(xor(self.L[i], shuffle(''.join(self.boxes[i]), self.p\_table)))

if self.debug:

print("="\*96+"\n")

print("Feistel\n")

for i in range(17):

print(f"L{i}\t\t: {split(self.L[i], 4)}")

print(f"R{i}\t\t: {split(self.R[i], 4)}")

print()

if i < 16:

print(f"#Round {i+1}")

print(f"E(R{i})\t\t: {split(self.new\_r[i], 6)}")

print(f"K{i+1}\t\t: {split(self.keys[i+1], 6)}")

print(f"K{i+1}+E(R{i})\t: {split(self.kr[i], 6)}")

print(f"S-Box\t\t: {' '.join(self.boxes[i])}")

print(f"S-Box-P\t\t: {split(shuffle(''.join(self.boxes[i]), self.p\_table), 4)}")

return self.R[16] + self.L[16]

class DES:

def \_\_init\_\_(self, key: str, debug=False):

self.key = hex\_to\_bin(key)

self.pc1 = (

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,

)

self.pc2 = (

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,

)

self.ip\_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,

)

self.shift\_table = (1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1)

self.reverse\_ip\_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

)

self.debug = debug

self.K = [shuffle(self.key, self.pc1)]

self.C = [self.K[0][:28]]

self.D = [self.K[0][28:]]

self.generate\_subkeys()

def encrypt(self, msg: str) -> str:

msg = hex\_to\_bin(msg)

print(f"msg: {split(msg, 8)}\nkey: {split(self.key, 8)}")

msg = shuffle(msg, self.ip\_table)

if self.debug:

print(f"shuffled msg: {split(msg, 8)}\n")

print("="\*96+"\n")

print(f"K0\t: {split(self.K[0], 7)}")

print(f"C0\t: {split(self.C[0], 7)}")

print(f"D0\t: {split(self.D[0], 7)}")

print()

for i in range(1, 17):

print(f"C{i}\t: {split(self.C[i], 7)}")

print(f"D{i}\t: {split(self.D[i], 7)}")

print(f"C{i}D{i}\t: {split(self.C[i]+self.D[i], 7)}")

print(f"K{i}\t: {split(self.K[i], 6)}")

print()

left = msg[:32]

right = msg[32:]

feistel = Feistel(left, right, self.K, xor, self.debug)

res = feistel.round16()

if self.debug:

print(f"R16L16\t\t: {split(res, 8)}")

res = shuffle(res, self.reverse\_ip\_table)

if self.debug:

print(f"Reverse IP\t: {split(res, 8)}")

print()

print("="\*96)

print()

return bin\_to\_hex(res)

def decrypt(self, ciphertext) -> str:

ciphertext = hex\_to\_bin(ciphertext)

ciphertext = shuffle(ciphertext, self.ip\_table)

left = ciphertext[:32]

right = ciphertext[32:]

feistel = Feistel(left, right, [self.K[0]]+self.K[1:][::-1], xor, self.debug)

res = feistel.round16()

res = shuffle(res, self.reverse\_ip\_table)

return bin\_to\_hex(res)

def generate\_subkeys(self) -> None:

for i in range(16):

shift = self.shift\_table[i]

self.C.append(self.C[i][shift:]+self.C[i][:shift])

self.D.append(self.D[i][shift:]+self.D[i][:shift])

for i in range(16):

self.K.append(shuffle(self.C[i+1] + self.D[i+1], self.pc2))

HEADER = 64

PORT = 5051

SERVER = socket.gethostbyname(socket.gethostname())

ADDR = (SERVER, PORT)

FORMAT = 'utf-8'

DISCONNECT\_MESSAGE = "!DISCONNECT"

key = "myChiper"

key = string2hex(key)

des = DES(key, False)

server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server.bind(ADDR)

def handle\_client(conn, addr):

print(f"[NEW CONNECTION] {addr} connected.")

connected = True

while connected:

msg\_length = conn.recv(HEADER).decode(FORMAT)

if msg\_length:

msg\_length = int(msg\_length)

msg = conn.recv(msg\_length).decode(FORMAT)

if msg == DISCONNECT\_MESSAGE:

connected = False

hex\_string = des.decrypt(msg).upper()

bytes\_object = bytes.fromhex(hex\_string)

ascii\_string = bytes\_object.decode("ASCII")

print(f"[{addr}] {ascii\_string}")

conn.send("Msg received\n".encode(FORMAT))

conn.close()

def start():

server.listen()

print(f"[LISTENING] Server is listening on {SERVER}")

while True:

conn, addr = server.accept()

thread = threading.Thread(target=handle\_client, args=(conn, addr))

thread.start()

print(f"[ACTIVE CONNECTIONS] {threading.activeCount() - 1}")

print("[STARTING] server is starting...")

start()

1. Client.py

import socket

from functools import reduce

def string2hex(s):

List=[]

for ch in s:

st2hx = hex(ord(ch)).replace('0x','')

if(len(st2hx)==1): st2hx = '0' + st2hx

List.append(st2hx)

return reduce(lambda i, j: i+j, List)

def xor(a: str, b: str) -> str:

return bin(int(a,2) ^ int(b,2))[2:].rjust(len(a), '0')

def split(msg: str, n: int) -> str:

return ' '.join(msg[i:i+n] for i in range(0, len(msg), n))

def shuffle(key: str, table: tuple) -> str:

return "".join(key[i-1] for i in table)

def hex\_to\_bin(h: str) -> str:

return "".join(map(lambda x: bin(int(x, 16))[2:].rjust(4, '0'), list(h)))

def bin\_to\_hex(b: str) -> str:

return "".join(map(lambda x: hex(int(x, 2))[2:], split(b, 4).split()))

class Feistel:

def \_\_init\_\_(self, left: str, right: str, keys: list, f, debug=True):

self.L = [left]

self.R = [right]

self.new\_r = []

self.kr = []

self.boxes = []

self.keys = keys

self.f = f

self.p\_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

)

self.e\_bit\_selection\_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

)

self.SBox = (

(

(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)

)

)

self.debug = debug

def round16(self) -> str:

for i in range(16):

self.L.append(self.R[i])

self.new\_r.append(shuffle(self.R[i], self.e\_bit\_selection\_table))

self.kr.append(self.f(self.new\_r[i], self.keys[i+1]))

self.boxes.append(split(self.kr[i], 6).split())

for idx, b in enumerate(self.boxes[-1]):

row = int(b[0]+b[5], 2)

col = int(b[1:5], 2)

self.boxes[-1][idx] = bin(self.SBox[idx][row][col])[2:].rjust(4, '0')

self.R.append(xor(self.L[i], shuffle(''.join(self.boxes[i]), self.p\_table)))

if self.debug:

print("="\*96+"\n")

print("Feistel\n")

for i in range(17):

print(f"L{i}\t\t: {split(self.L[i], 4)}")

print(f"R{i}\t\t: {split(self.R[i], 4)}")

print()

if i < 16:

print(f"#Round {i+1}")

print(f"E(R{i})\t\t: {split(self.new\_r[i], 6)}")

print(f"K{i+1}\t\t: {split(self.keys[i+1], 6)}")

print(f"K{i+1}+E(R{i})\t: {split(self.kr[i], 6)}")

print(f"S-Box\t\t: {' '.join(self.boxes[i])}")

print(f"S-Box-P\t\t: {split(shuffle(''.join(self.boxes[i]), self.p\_table), 4)}")

return self.R[16] + self.L[16]

class DES:

def \_\_init\_\_(self, key: str, debug=False):

self.key = hex\_to\_bin(key)

self.pc1 = (

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,

)

self.pc2 = (

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,

)

self.ip\_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,

)

self.shift\_table = (1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1)

self.reverse\_ip\_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

)

self.debug = debug

self.K = [shuffle(self.key, self.pc1)]

self.C = [self.K[0][:28]]

self.D = [self.K[0][28:]]

self.generate\_subkeys()

def encrypt(self, msg: str) -> str:

msg = hex\_to\_bin(msg)

# print(f"msg: {split(msg, 8)}\nkey: {split(self.key, 8)}")

msg = shuffle(msg, self.ip\_table)

if self.debug:

print(f"shuffled msg: {split(msg, 8)}\n")

print("="\*96+"\n")

print(f"K0\t: {split(self.K[0], 7)}")

print(f"C0\t: {split(self.C[0], 7)}")

print(f"D0\t: {split(self.D[0], 7)}")

print()

for i in range(1, 17):

print(f"C{i}\t: {split(self.C[i], 7)}")

print(f"D{i}\t: {split(self.D[i], 7)}")

print(f"C{i}D{i}\t: {split(self.C[i]+self.D[i], 7)}")

print(f"K{i}\t: {split(self.K[i], 6)}")

print()

left = msg[:32]

right = msg[32:]

feistel = Feistel(left, right, self.K, xor, self.debug)

res = feistel.round16()

if self.debug:

print(f"R16L16\t\t: {split(res, 8)}")

res = shuffle(res, self.reverse\_ip\_table)

if self.debug:

print(f"Reverse IP\t: {split(res, 8)}")

print()

print("="\*96)

print()

return bin\_to\_hex(res)

def decrypt(self, ciphertext) -> str:

ciphertext = hex\_to\_bin(ciphertext)

ciphertext = shuffle(ciphertext, self.ip\_table)

left = ciphertext[:32]

right = ciphertext[32:]

feistel = Feistel(left, right, [self.K[0]]+self.K[1:][::-1], xor, self.debug)

res = feistel.round16()

res = shuffle(res, self.reverse\_ip\_table)

return bin\_to\_hex(res)

def generate\_subkeys(self) -> None:

for i in range(16):

shift = self.shift\_table[i]

self.C.append(self.C[i][shift:]+self.C[i][:shift])

self.D.append(self.D[i][shift:]+self.D[i][:shift])

for i in range(16):

self.K.append(shuffle(self.C[i+1] + self.D[i+1], self.pc2))

# msg = "0123456712332332"

HEADER = 64

PORT = 5051

FORMAT = 'utf-8'

DISCONNECT\_MESSAGE = "!DISCONNECT"

SERVER = socket.gethostbyname(socket.gethostname())

ADDR = (SERVER, PORT)

client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client.connect(ADDR)

def send(msg):

message = msg.encode(FORMAT)

msg\_length = len(message)

send\_length = str(msg\_length).encode(FORMAT)

send\_length += b' ' \* (HEADER - len(send\_length))

client.send(send\_length)

client.send(message)

print(client.recv(2048).decode(FORMAT))

def pad(text):

while len(text) %8 != 0:

text += ' '

return text

x = ' '

while x != 'quit':

# Ask the user for a name.

key = "myChiper"

x = ' '

x = input()

pad\_text = pad(x)

msg = string2hex(pad\_text)

key = string2hex(key)

des = DES(key, False)

enc = des.encrypt(msg).upper()

print(f"encrypted: {enc}")

send(enc)

send(DISCONNECT\_MESSAGE)

Hasil run

