**LAB\_01.02.2025**

1.Write a program for Hill cipher succumbs to a known plaintext attack if sufficient plaintext–

ciphertext pairs are provided. It is even easier to solve the Hill cipher if a chosen plaintext attack

can be mounted.

def mi(n, mod):

for i in range(1, mod):

if (n \* i) % mod == 1:

return i

return -1

def rk(pt, ct, mod=26):

d = (pt[0][0] \* pt[1][1] - pt[0][1] \* pt[1][0]) % mod

di = mi(d, mod)

if di == -1:

raise ValueError("Matrix is not invertible under the given modulus")

pi = [[pt[1][1] \* di % mod, -pt[0][1] \* di % mod],

[-pt[1][0] \* di % mod, pt[0][0] \* di % mod]]

k = [[sum(a \* b for a, b in zip(row, col)) % mod for col in zip(\*pi)] for row in ct]

return k

pt = [[7, 8], [4, 19]]

ct = [[19, 3], [4, 21]]

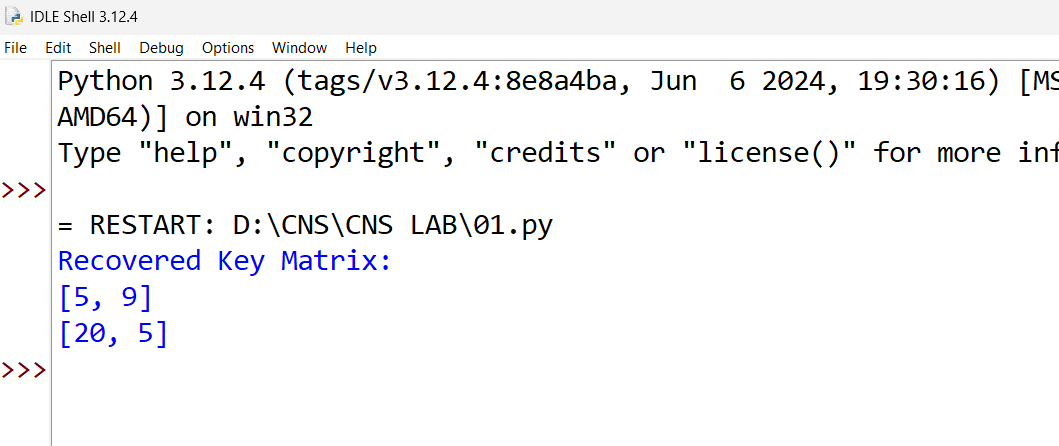
km = rk(pt, ct)

print("Recovered Key Matrix:")

for row in km:

print(row)

output:



2. Write a program that can perform a letter frequency attack on an additive cipher without

human intervention. Your software should produce possible plaintexts in rough order of

likelihood. It would be good if your user interface allowed the user to specify “give me the top

10 possible plaintexts.”

f = "ETAOINSHRDLCUMWFGYPBVKJXQZ"

def decr(ct, k):

return "".join(chr(((ord(c) - 65 - k) % 26) + 65) if c.isalpha() else c for c in ct)

def score(txt):

return sum(abs(f.index(c) - i) for i, c in enumerate(sorted(set(txt), key=txt.count, reverse=True)) if c in f)

def attack(ct, n=10):

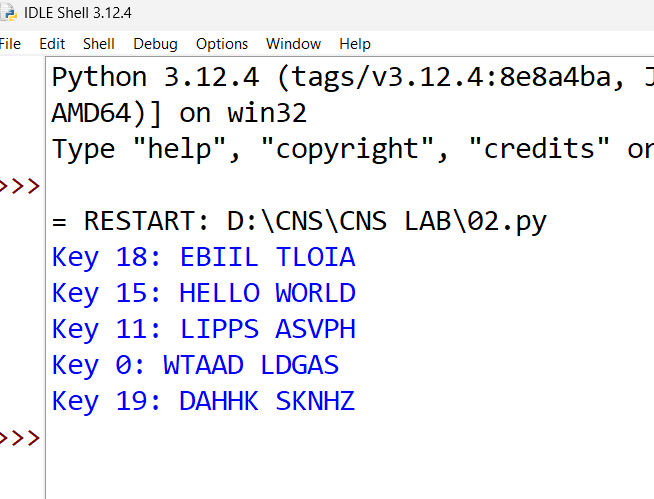
return sorted([(k, decr(ct, k)) for k in range(26)], key=lambda x: score(x[1]))[:n]

ct = "WTAAD LDGAS"

for k, txt in attack(ct, 5):

print(f"Key {k}: {txt}")

output:



3. Write a program for DES algorithm for decryption, the 16 keys (K1, K2, c, K16) are used in

reverse order. Design a key-generation scheme with the appropriate shift schedule for the

decryption process.

ip = [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]

ipinv = [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]

e = [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]

p = [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]

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]

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]

shifts = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]

def permute(x, table):

return ''.join(x[i-1] for i in table)

def shift(x, s):

return x[s:] + x[:s]

def gkey(k):

k = permute(k, pc1)

l, r = k[:28], k[28:]

keys = []

for s in shifts:

l, r = shift(l, s), shift(r, s)

keys.append(permute(l + r, pc2))

return keys[::-1]

def f(r, k):

r = permute(r, e)

r = bin(int(r, 2) ^ int(k, 2))[2:].zfill(48)

return permute(r, p)

def d(ct, k):

ct = permute(ct, ip)

l, r = ct[:32], ct[32:]

keys = gkey(k)

for i in range(16):

l, r = r, bin(int(l, 2) ^ int(f(r, keys[i]), 2))[2:].zfill(32)

return permute(r + l, ipinv)

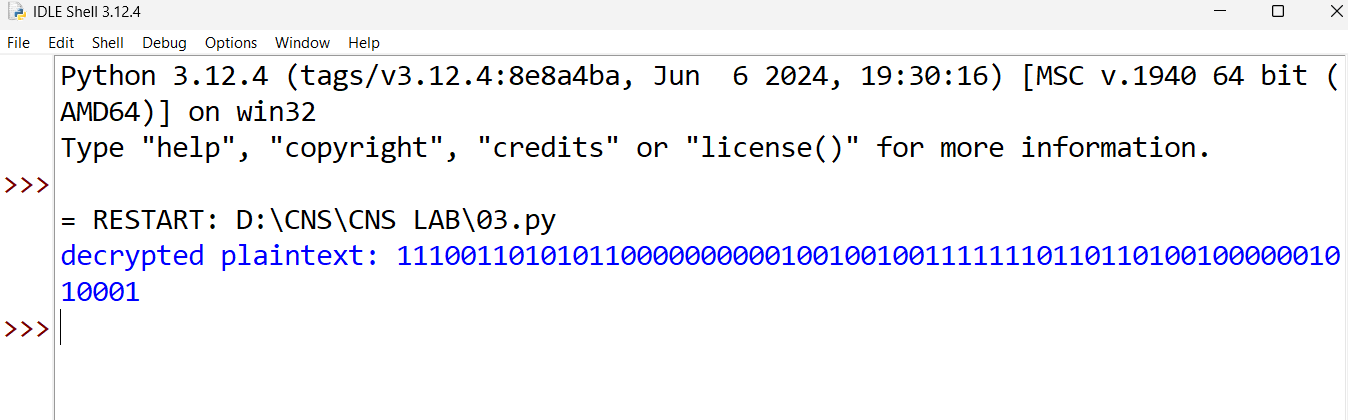
ct = "1100110000000000110011001111111111001100000000001100110011111111"

k = "0001001100110100010101110111100110011011101111001101111111110001"

pt = d(ct, k)

print("decrypted plaintext:", pt)

output:



4. Write a program for DES the first 24 bits of each subkey come from the same subset of 28 bits

of the initial key and that the second 24 bits of each subkey come from a disjoint subset of 28

bits of the initial key.

Code:

ip = [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]

fip = [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]

e = [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]

p = [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]

pcone = [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]

shifts = [1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1]

def permute(x, t):

return ''.join(x[i-1] for i in t)

def shift(x, s):

return x[s:] + x[:s]

def gkey(k):

k = permute(k, pcone)

l = k[:28]

r = k[28:]

ks = []

for s in shifts:

l = shift(l, s)

r = shift(r, s)

sk = l[:24] + r[:24]

ks.append(sk)

return ks

def f(r, k):

x = permute(r, e)

x = int(x, 2) ^ int(k, 2)

x = bin(x)[2:].zfill(48)

return permute(x, p)

def enc(pt, k):

pt = permute(pt, ip)

l = pt[:32]

r = pt[32:]

ks = gkey(k)

for sk in ks:

temp = r

r = bin(int(l,2) ^ int(f(r, sk),2))[2:].zfill(32)

l = temp

ct = permute(r + l, fip)

return ct

def dec(ct, k):

ct = permute(ct, ip)

l = ct[:32]

r = ct[32:]

ks = gkey(k)[::-1]

for sk in ks:

temp = r

r = bin(int(l,2) ^ int(f(r, sk),2))[2:].zfill(32)

l = temp

pt = permute(r + l, fip)

return pt

pt = "0000000100100011010001010110011110001001101010111100110111101111"

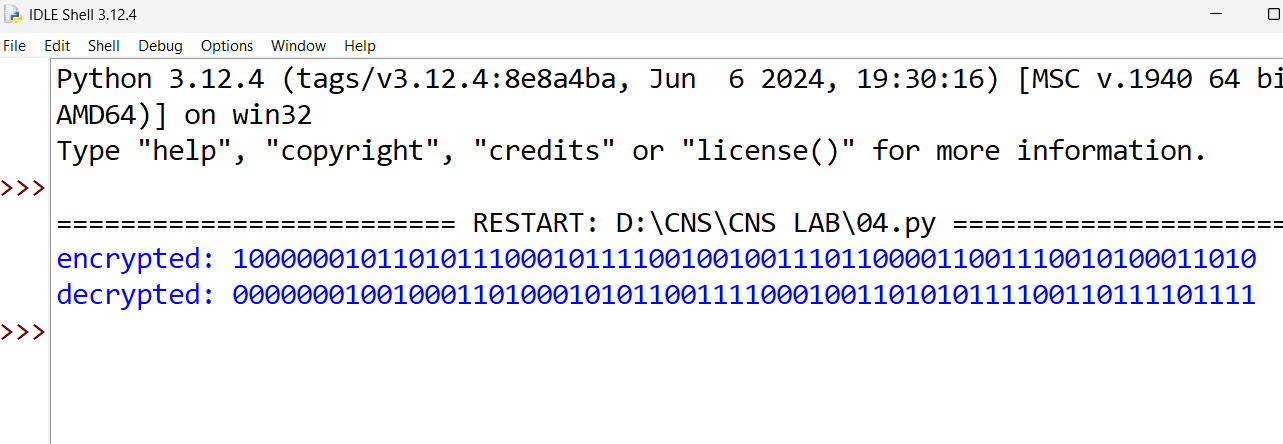
k = "0001001100110100010101110111100110011011101111001101111111110001"

ct = enc(pt, k)

print("encrypted:", ct)

print("decrypted:", dec(ct, k))

output:



5. Write a program for encryption in the cipher block chaining (CBC) mode using an algorithm

stronger than DES. 3DES is a good candidate. Both of which follow from the definition of CBC.

Which of the two would you choose:

a. For security?

b. For performance?

Code:

ip = [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]

fip = [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]

e = [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]

p = [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]

pcone = [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]

shifts = [1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1]

def pmt(x, t):

s = ''

for i in t:

s = s + x[i-1]

return s

def sft(x, n):

return x[n:] + x[:n]

def gk(k):

k = pmt(k, pcone)

l = k[0:28]

r = k[28:]

ks = []

for n in shifts:

l = sft(l, n)

r = sft(r, n)

sk = l[0:24] + r[0:24]

ks.append(sk)

return ks

def ff(r, k):

x = pmt(r, e)

x = int(x, 2) ^ int(k, 2)

x = bin(x)[2:].zfill(48)

return pmt(x, p)

def enc(pt, k):

pt = pmt(pt, ip)

l = pt[0:32]

r = pt[32:]

ks = gk(k)

for sk in ks:

t = r

r = bin(int(l,2) ^ int(ff(r, sk),2))[2:].zfill(32)

l = t

ct = pmt(r + l, fip)

return ct

def dec(ct, k):

ct = pmt(ct, ip)

l = ct[0:32]

r = ct[32:]

ks = gk(k)

ks = ks[::-1]

for sk in ks:

t = r

r = bin(int(l,2) ^ int(ff(r, sk),2))[2:].zfill(32)

l = t

pt = pmt(r + l, fip)

return pt

def tde(pt, k1, k2, k3):

t = enc(pt, k3)

t = dec(t, k2)

ct = enc(t, k1)

return ct

def xor(a, b):

s = ''

for i in range(len(a)):

if a[i] == b[i]:

s = s + '0'

else:

s = s + '1'

return s

def tdescbc(txt, iv, k1, k2, k3):

n = 64

ct = ''

prev = iv

for i in range(0, len(txt), n):

blk = txt[i:i+n]

blk = xor(blk, prev)

cblk = tde(blk, k1, k2, k3)

ct = ct + cblk

prev = cblk

return ct

pt = ("0000000100100011010001010110011110001001101010111100110111101111" +

"1111000011110000111100001111000011110000111100001111000011110000")

iv = "1010101010101010101010101010101010101010101010101010101010101010"

k1 = "0001001100110100010101110111100110011011101111001101111111110001"

k2 = "0101010101010101010101010101010101010101010101010101010101010101"

k3 = "0011001100110011001100110011001100110011001100110011001100110011"

ct = tdescbc(pt, iv, k1, k2, k3)

print("encrypted:", ct)

print("for security: triple des")

print("for performance: des")

Output:

