University of Birmingham - Cryptography DES Exercise 1

Haotian Xiang

November 9, 2019

1 Exercise 1

1.1 Questions

1.1.1 Question 1

As you know, the rail fence cipher is weak, because you can exhaustively try all the keys. Find the plaintext for the the following ciphertext encrypted with the rail fence cipher: AVUEVLETSEISBNACBOOLEOBTILBDLCOBOOE

1.1.2 Answer 1

Code:

```
encrypted_string = "AVUEVLETSEISBNACBOOLEOBTILBDLCOBOOE"

def decrypt(encrypted_string, key):
    decrypted_string = ""
    key = int(len(encrypted_string) / key)
    for y in range(0, key):
        for char in range(y, len(encrypted_string), key):
            decrypted_string += encrypted_string[char]
    return decrypted_string

for key in range(1, divmod(encrypted_string.__len__(), 2)[0]):
    print(decrypt(encrypted_string, key))


Output:
```

AVUEVLETSEISBNACBOOLEOBTILBDLCOBOOE
AOEVOULEEVOLBETTISLEBIDSLBCNOABCOBO
ASBOVBTEUNIEALVCBLBDEOLTOCSLOEEBIOO

ASBIOVEOLOUIOBEESLDVBELLNOCEABOTCTB
ATAOLVSCBCUEBTOEIOIBVSOLOLBLBOENEDE
ALICELOVESBOBBUTBOBDOESNOTLOVEALICE
ALICELOVESBOBBUTBOBDOESNOTLOVEALICE
AVSBBEILOVLENOOLCOUEIAOBBOEETSCLTDB
AEEEBCOOIDOOVVTINBLBLLBEULSSAOETBCO
AEEEBCOOIDOOVVTINBLBLLBEULSSAOETBCO
AEEEBCOOIDOOVVTINBLBLLBEULSSAOETBCO
AUVESIBABOEBIBLOOEVELTESNCOLOTLDCBO
AUVESIBABOEBIBLOOEVELTESNCOLOTLDCBO
AUVESIBABOEBIBLOOEVELTESNCOLOTLDCBO
AUVESIBABOEBIBLOOEVELTESNCOLOTLDCBO
AUVESIBABOEBIBLOOEVELTESNCOLOTLDCBO

Answer is :
ALICELOVESBOBBUTBOBDOESNOTLOVEALICE

1.1.3 Question 2

Assume a simple two-round Feistel block cipher with an 8 bit key and a 16 bit block size. We write the key as a decimal number (from 0 to 255) and the input as two decimal numbers (also from 0 to 255). The key derivation is defined as $K_i = K + 75 * i (mod 256)$
.Where $0 \leq i \leq 1. f(Ki, Ri) = 127 * (Ki + Ri)(mod 256)$. Where $Ri$is the decimal representation of the right 8 bits of the input block. Encrypt the message (86, 83) with the key 89.

1.1.4 Answer 2

Answer:
First round
(86,83)
$\because Ki = K + 75 * i(mod256)(0 \leq i \leq 1)$
if i = 0
$K_0 = K$
if i=1
$K_1 = K + 75(mod256)$
$\because key = 89$
$dec(86) = b"01010110"$
$dec(83) = b"01010011"$
$dec(89) = b"01011001"$
$L_0 = 86 = b"01010110"$
$R_0 = 83 = b"01010011"$
$\because K_i = K + 75 * i(mod256) 0 \leq i \leq 1.$
$\therefore K_0 = 89(mod256) = 89 = b"01011001"$
$\therefore K_1 = (89 + 75)(mod256) = 164(mod256) = 164 = b"10100100"$
$\because f(Ki, Ri) = 127 * (Ki + Ri)(mod256)$

$\therefore f(K_0, R_0) = 127 * (83 + 89)(mod 256) = 21844(mod 256) = 84.$
$\therefore R_1 = L_0 \oplus 84 = 86 \oplus 84 = 2$
$\therefore L_1 = R_0 = 83$
Second round
(83,2)
Same steps
$R_2 = L_1 \oplus (127 * (164 + 3))(mod 256) = 83 \oplus 90 = 9$
Final answer is (9,2)

1.1.5 Question 3

What is the output of the first round of the DES algorithm when the plain- text and the key are both all zeros?

1.1.6 Answer 3

Answer:
$\because$ plain text = 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
$\because$ key = 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
Key Generation:
1. Parity drop
$\therefore$ 56 bits key = 00000000 00000000 00000000 00000000 00000000 00000000 00000000
$\therefore$ all round key are 56bits.
after key schedule all keys remain all zero but they are turned to 48 bits.
$\therefore Key_0$ = 00000000 00000000 00000000 00000000 00000000 00000000
set first chunk to all zeros.
plain text is divided to two parts each part is 32bits.
Initial Permutation:
L_0 = 00000000000000000000000000000000 and R_0 = 00000000000000000000000000000000
First round
$\because M = (L_0, R_0)$
$\because L_1 = R_0$
$\because R_1 = L_0 \oplus F(R_0, K_1)$
Feistel function:
$\because$Expansion permutation
R_0 = 00000000 00000000 00000000 00000000 00000000 00000000
XOR:
R_0 = 00000000 00000000 00000000 00000000 00000000 00000000 $\oplus Key_1$ = 00000000 00000000 00000000 00000000 00000000 00000000
after S-Box:
1110 1111 1010 0111 0010 1100 0100 1101
after straight Permutation:
1101 1000 1101 1000 1101 1011 1011 1100

$\therefore R_1 = L_0 \oplus F(R_0, K_1) = 1110\ 1111\ 1010\ 0111\ 0010\ 1100\ 0100\ 1101$
$\therefore L_1 = 00000000\ 00000000\ 00000000\ 00000000$
$\therefore$ Final Answer is
00000000 00000000 00000000 00000000 11011000 11011000 11011011 10111100

1.1.7 Question 4

Remember that it is desirable for good block ciphers that a change in one input bit affects many output bits, a property that is called diffusion or avalanche effect. We will try to get a feeling for the avalanche property of DES. Let x be all zeros (0x0000000000000000) and y be all zeros except 1 in the 13th bit (0x0008000000000000). Let the key be all zeros. After just one round, how many bits in the block are different when x is the input, compared to when y is the input? What about after two rounds? Three? Four? (For this exercise, you might like to search for an implementation of DES on the web, and download it and modify it to output the answers.)

1.1.8 Answer 4

```
import hashlib
import string
import random
from random import randint
from sys import getsizeof
import datetime
import time
import threading
import os




hash1_table = dict()
hash2_table = dict()

def unique_strings(k: int, hash_number: int,
                   pool: str='1234567890ZAQWSXCDERFVBGTYHNJUIMKLOPzaqwsxcderfvb
    # An optimization for tightly-bound loops:
    # Bind these methods outside of a loop

    join = ''.join

    # while len(hash_table) < hash_number:
    token = join(random.choices(pool, k=k))
    return token
```

```python
def add_hash(tabel_number, table):
    while(len(table) != 10000000):
    # while(len(table) != 100000):
        random_string = unique_strings(randint(5,10),5)
        random_string_hash = hashlib.sha1(random_string.encode('utf-8')).hexd
        table.update({random_string_hash:random_string})
        print("table" + str(tabel_number) + ":" + str(len(table)))

def compare(first, second):
    sharedKeys = set(first.keys()).intersection(second.keys())
    for key in sharedKeys:
        print("comparing now")
        if first[key] != second[key]:
            print('Key: {}, Value 1: {}, Value 2: {}'.format(key, first[key],
            os._exit(1)

while True:
    compare(hash1_table,hash2_table)
    hash1_table = {}
    hash2_table = {}
    add_hash(1,hash1_table)
    add_hash(2,hash2_table)
```

Final answer:
if x is all zeros:
Four rounds:
0000000000000000000000000000000011011000110110001101101110111100
1101100011011000110110111011110011100111001110101110110101001111
1110011100111010111011010100111101011011111110100110101110100110
0101101111111010011010111010011001011110010001011110001001010111
if y is all zeros except 13th is 1:
Four rounds:
0000000000000000000100000000000011011000111110001101101110110100
1101100011111000110110111011010011100101001010001110001001101011
1110010100101000111000100110101110000111110101110000001000110110
1000011111010111000000100011011000111101000101001111010100111100