Cryptography module, Exercises 1 (unassessed) - Answers

All the code contained in this exercise solution is hosted at: https://github.com/amar-laksh/UNI/assignments/CRYPTO/code

Answer - 1:

Listing 1 contains python code to brute-force the given cipher-text ”*AVUEVLETSEISBNACBOOLEOBTILBDLCOBOOE*”.

Listing 1: ex1.1.py

```
from math import ceil

def encode(msg, key):
    cipher = ””
    for rails in range(0, key):
        for char in range(rails, len(msg), key):
            cipher += msg[char]
    return cipher

def decode(cipher, key):
    msg = ””
    count = ceil(len(cipher)/key)
    for rails in range(0, count):
        for c in range(rails, len(cipher), count):
            msg += cipher[c]
    return msg

def crack_cipher(cipher):
    for key in range(1, len(cipher)):
        print(decode(cipher, key))

cipher = ”AVUEVLETSEISBNACBOOLEOBTILBDLCOBOOE”

crack_cipher(cipher)
```

Listing 2 contains the output of the python code in **Listing 1** containing the message, ”*ALICELOVESBOBBUTBOBDOESNOTLOVEALICE*” shown in the bold.

Listing 2: output of ex1.1.py

```
AVUEVLETSEISBNACBOOLEOBTILBDLCOBOOE
AOVLUEEOVBLTEITLSBEDILSCBONBAOCOBEO
ABIVNLUABECDVBLLOCEOOTLBSEOEOOIBEST
AEODVILLUSECEBOOVNBBLATOECIOTBLESOB
ATAOLVSCBCUEBTOEIOIBVSOLOLBLBOENEDE
AEBOIOVTNLLBUSAEBOEECODOVIBBLELSOTC
ALICELOVESBOBBUTBOBDOESNOTLOVEALICE
ALICELOVESBOBBUTBOBDOESNOTLOVEALICE
AVSBBEILOVLENOOLCOUEIAOBBOEETSCLTDB
AVSBBEILOVLENOOLCOUEIAOBBOEETSCLTDB
AVSBBEILOVLENOOLCOUEIAOBBOEETSCLTDB
AEEEBCOOIDOOVVTINBLBLLBEULSSAOETBCO
AEEEBCOOIDOOVVTINBLBLLBEULSSAOETBCO
AEEEBCOOIDOOVVTINBLBLLBEULSSAOETBCO
```

```
AEEEBCOOIDOOVVTINBLBLLBEULSSAOETBCO
AEEEBCOOIDOOVVTINBLBLLBEULSSAOETBCO
AEEEBCOOIDOOVVTINBLBLLBEULSSAOETBCO
AUVESIBABOEBIBLOOEVELTESNCOLOTLDCBO
AUVESIBABOEBIBLOOEVELTESNCOLOTLDCBO
AUVESIBABOEBIBLOOEVELTESNCOLOTLDCBO
AUVESIBABOEBIBLOOEVELTESNCOLOTLDCBO
AUVESIBABOEBIBLOOEVELTESNCOLOTLDCBO
AUVESIBABOEBIBLOOEVELTESNCOLOTLDCBO
AUVESIBABOEBIBLOOEVELTESNCOLOTLDCBO
AUVESIBABOEBIBLOOEVELTESNCOLOTLDCBO
AUVESIBABOEBIBLOOEVELTESNCOLOTLDCBO
AUVESIBABOEBIBLOOEVELTESNCOLOTLDCBO
AUVESIBABOEBIBLOOEVELTESNCOLOTLDCBO
AUVESIBABOEBIBLOOEVELTESNCOLOTLDCBO
AUVESIBABOEBIBLOOEVELTESNCOLOTLDCBO
AUVESIBABOEBIBLOOEVELTESNCOLOTLDCBO
AUVESIBABOEBIBLOOEVELTESNCOLOTLDCBO
AUVESIBABOEBIBLOOEVELTESNCOLOTLDCBO
AUVESIBABOEBIBLOOEVELTESNCOLOTLDCBO
```

Answer - 2:

Listing 3 contains the python code to perform encryption and decryption using the block cipher scheme mentioned in the question.

Listing 3: ex1.2.py

```python
ROUNDS = 2


def key_function(K, i):
    return K + 75 * (i % 256)


def F(Ki, Pi):
    return 127 * Ki + (Pi % 256)


def encrypt(msg, key):
    Li = msg[0]
    Ri = msg[1]
    temp = 0
    for i in range(0, ROUNDS):
        Ki = key_function(key, i)
        temp = Li ^ F(Ki, Ri)
        Li = Ri
        Ri = temp
    return [Ri, Li]


def decrypt(cipher, key):
    Li = cipher[1]
    Ri = cipher[0]
    temp = 0
    for i in range(ROUNDS, 0, -1):
        Ki = key_function(key, (i - 1))
        temp = Ri ^ F(Ki, Li)
        Ri = Li
        Li = temp
    return [Li, Ri]


print(encrypt([86, 83], 89))
```

Listing 4 contains the output of the python code in **Listing 3** containing the encrypted cipher-text ”*[20955, 11308]*”.

Listing 4: output of ex1.2.py

```
[20955, 11308]
```

Answer - 3:

Listing 5 contains the python code to perform encryption using a modified version of the DES implementation in Python.

The modified fork of the DES python library can be found at: https://github.com/amar-laksh/pydes

Listing 5: ex1.3.py

```
import sys
sys.path.insert(1, 'pydes')
from pydes import des


def h2b(byte_string):
    return bytes.fromhex(byte_string)


key = h2b("0000000000000000")
msg = h2b("0000000000000000")

d = des(rounds=1)
cipher = d.encrypt(key, msg)

cipher = '␣'.join(format(ord(x), 'b') for x in cipher)

print("The␣ciphertext␣after␣1␣round:␣%s" % cipher)
```

Listing 6 contains the output of the python code in **Listing 5** containing the encrypted cipher-text ”*100 100 1 1010101 1010101 1 1010100 1010101*”.

Listing 6: output of ex1.3.py

```
The ciphertext after 1 round: 100 100 1 1010101 1010101 1 1010100 1010101
```

This cipher-text is obtained even when the encryption key and plain-text is all zeroes.

Answer - 4:

Listing 7 contains the python code to perform encryption using a modified version of the DES implementation in Python.

The modified fork of the DES python library can be found at: https://github.com/amar-laksh/pydes

Listing 7: ex1.4.py

```
import sys
sys.path.insert(1, ’pydes’)
from pydes import des


def h2b(byte_string):
    return bytes.fromhex(byte_string)


key = h2b(”0000000000000000”)
x = h2b(”0000000000000000”)
y = h2b(”0008000000000000”)


for i in range(1, 17):
    print(”ROUND␣%i:” % i)

    d = des(rounds=i)
    cipher_x = d.encrypt(key, x)
    cipher_y = d.encrypt(key, y)

    cipher_x = int.from_bytes(bytes(cipher_x, ”UTF−8”), ”little”)
    cipher_y = int.from_bytes(bytes(cipher_y, ”UTF−8”), ”little”)

    # XOR operation can be used to get the number of different bits
    diff_x_y = ((cipher_x) ^ (cipher_y))

    print(hex(cipher_x))
    print(hex(cipher_y))
    print(”NUMBER␣OF␣BITS␣DIFFERENT:␣%s” % bin(diff_x_y).count(’1’))
```

Listing 8 contains the output of the python code in **Listing 7** containing the encrypted cipher-text and the number of bits different when two inputs are ”x” and ”y” respectively.

Listing 8: output of ex1.4.py

```
ROUND 1:
0x5554015555010404
0x5554115554010c04
NUMBER OF BITS DIFFERENT: 3
ROUND 2:
0xaec3adc356bac2bfc247594d
0xaec3adc377aac2b9c2420d49
NUMBER OF BITS DIFFERENT: 11
ROUND 3:
0x99c29ec3bdc2707e8bc2b7c38ec3
0x98c39ac2abc2112291c35f92c3
NUMBER OF BITS DIFFERENT: 57
ROUND 4:
0x26bdc32ea1c3a8c353afc399c2
0xa4c224477741b7c3aac2a4c3
NUMBER OF BITS DIFFERENT: 38
ROUND 5:
0x49afc20d96c294c3a2c28ec376
0x5c4d8ac3bbc386c3bac34589c2
NUMBER OF BITS DIFFERENT: 36
ROUND 6:
0x96c35b5a68bdc34089c2bcc3
0xb9c29fc390c2b6c39dc3b5c29bc252
NUMBER OF BITS DIFFERENT: 60
```

```
ROUND 7:
 0xb8c2b3c2a0c291c3bfc391c216bcc3
 0x77afc271bcc3afc26e63b5c3
NUMBER OF BITS DIFFERENT: 55
ROUND 8:
 0x613311b7c3bac2223cacc2
 0xbfc30eb2c2b8c31b89c286c3aec3
NUMBER OF BITS DIFFERENT: 53
ROUND 9:
 0x82c32322bbc274042818
 0xafc20821b4c327178dc29cc2
NUMBER OF BITS DIFFERENT: 53
ROUND 10:
 0x90c2174137a9c2490074
 0x1a0457b8c25a3b4e2c
NUMBER OF BITS DIFFERENT: 36
ROUND 11:
 0x747b93c36f1393c251bcc3
 0x600cbec330a4c23298c349
NUMBER OF BITS DIFFERENT: 45
ROUND 12:
 0xbcc3b2c2b2c28fc32266b7c3bcc2
 0x80c35cbcc2250c61b5c382c3
NUMBER OF BITS DIFFERENT: 48
ROUND 13:
 0xacc271249ac35198c2afc33c
 0x84c3bdc23c1f4d97c3bbc280c3
NUMBER OF BITS DIFFERENT: 59
ROUND 14:
 0x1ca2c20ca0c2b7c2748bc26c
 0x99c27f392b8ec2bbc32381c3
NUMBER OF BITS DIFFERENT: 50
ROUND 15:
 0x791158407eacc2538cc3
 0x72bfc326174ca6c34296c3
NUMBER OF BITS DIFFERENT: 34
ROUND 16:
 0xa7c223b1c281c3a9c34da6c28cc2
 0xa1c2abc35d6e8dc28cc390c3adc2
NUMBER OF BITS DIFFERENT: 46
```