VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

CRYPTOGRAPHY AND NETWORK SECURITY
AAT REPORT
on

BLOWFISH ALGORITHM

Submitted by

REVANTH.S(1BM19CS128)
RISHANTH.K(1BM19CS129)

Under the Guidance of
Prof. Lohith J

Assistant Professor, BMSCE

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B. M. S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
Mar-2022 to Jun-2022

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the AAT work entitled “**BLOWFISH ALGORITHM**" is carried out by **REVANTH.S(1BM19CS128), and RISHANTH.K(1BM19CS129)** who are bonafide students of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022. The AAT report has been approved as it satisfies the academic requirements in respect of **Cryptography and Network Security (20CS6PCCNS)** work prescribed for the said degree.

Signature of the Guide	Signature of the HOD
Prof. LOHITH J	Dr. JYOTHI S. NAYAK
Assistant Professor	Associate Prof. & Head, Dept. of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

B. M. S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We,REVANTH.S(1BM19CS128) and RISHANTH.K(1BM19CS129), students of 6th Semester, B.E, Department of Computer Science and Engineering, B. M. S. College of Engineering, Bangalore, hereby declare that, this AAT entitled "BLOWFISH ALGORITHM" has been carried out by us under the guidance of Prof. Lohith J, Assistant Professor, Department of CSE, B. M. S. College of Engineering, Bangalore during the academic semester Mar-2022-Jun-2022

We also declare that to the best of our knowledge and belief, the development reported here is not from part of any other report by any other students.

Signature

REVANTH.S(1BM19CS128)

RISHANTH.K(1BM19CS128)

Chapter 1

Introduction

Blowfish is an encryption technique designed by **Bruce Schneier** in 1993 as an alternative to DES Encryption Technique. It is significantly faster than DES and provides a good encryption rate with no effective cryptanalysis technique found to date. It is one of the first, secure block cyphers not subject to any patents and hence freely available for anyone to use.

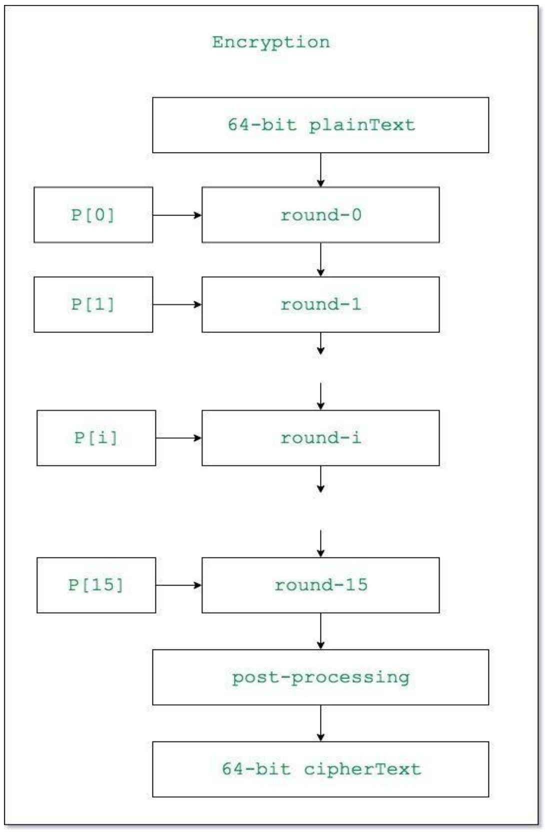
1. **BlockSize**: 64-bits
2. **KeySize**: 32-bits to 448-bits variable size
3. **Number of subkeys**: 18 [P-array]
4. **Number of rounds**: 16
5. **Number of substitution boxes**: 4 [each having 512 entries of 32-bits each]

Chapter 2

Methodology

Blowfish Encryption Algorithm

The entire encryption process can be elaborated as:



Step1: Generation of subkeys:

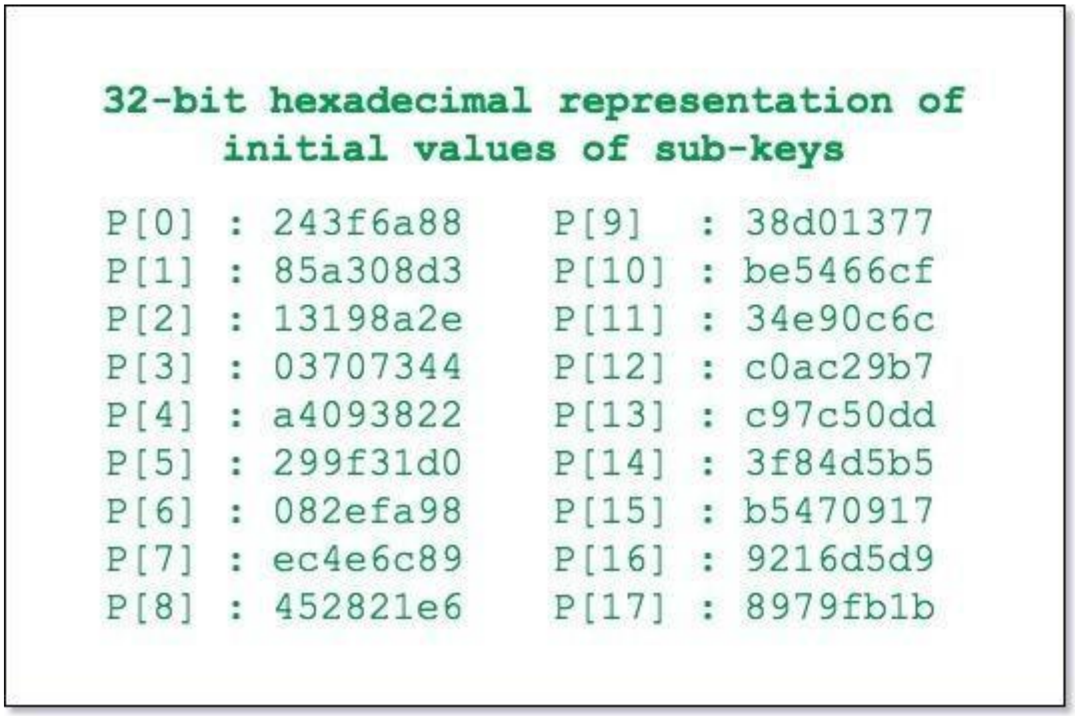
- 18 subkeys{P[0]…P[17]} are needed in both encryption as well as decryption process and the same subkeys are used for both the processes.
- These 18 subkeys are stored in a P-array with each array element being a 32-bit entry.
- It is initialised with the digits of pi(?).

The hexadecimal representation of each of the subkeys is given by:
P[0] = "243f6a88"

P[1] = "85a308d3"

.

P[17] = "8979fb1b"



32-bit hexadecimal representation of initial values of sub-keys

P[0]	243f6a88	P[9]	38d01377
P[1]	85a308d3	P[10]	be5466cf
P[2]	13198a2e	P[11]	34e90c6c
P[3]	03707344	P[12]	c0ac29b7
P[4]	a4093822	P[13]	c97c50dd
P[5]	299f31d0	P[14]	3f84d5b5
P[6]	082efa98	P[15]	b5470917
P[7]	ec4e6c89	P[16]	9216d5d9
P[8]	452821e6	P[17]	8979fb1b

Now each of the subkey is changed with respect to the input key as:
P[0] = P[0] xor 1st 32-bits of input key

P[1] = P[1] xor 2nd 32-bits of input key

.

P[i] = P[i] xor (i+1)th 32-bits of input key

(roll over to 1st 32-bits depending on the key length)

.

P[17] = P[17] xor 18th 32-bits of input key

(roll over to 1st 32-bits depending on key length)

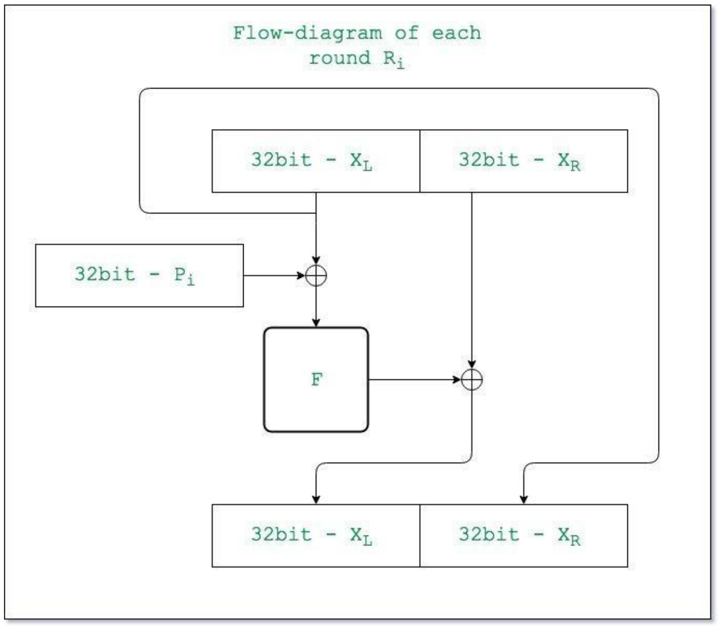
- The resultant P-array holds 18 subkeys that are used during the entire encryption process.

Step2: Initialise Substitution Boxes:

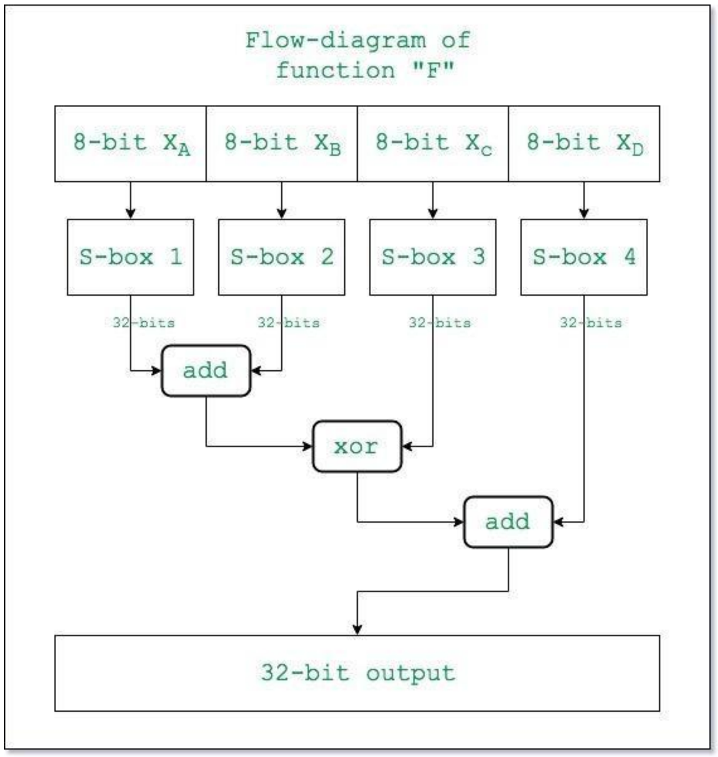
- 4 Substitution boxes(S-boxes) are needed{S[0]…S[4]} in both encryption as well as decryption process with each S-box having 256 entries{S[i][0]…S[i][255], 0&lei&le4} where each entry is 32-bit.
- It is initialised with the digits of pi(?) after initialising the P-array. You may find the s-boxes in here!

Step3: Encryption:

- The encryption function consists of two parts:
  1. **Rounds:** The encryption consists of 16 rounds with each round(Ri) taking inputs the plainText(P.T.) from the previous round and corresponding subkey(Pi). The description of each round is as follows:

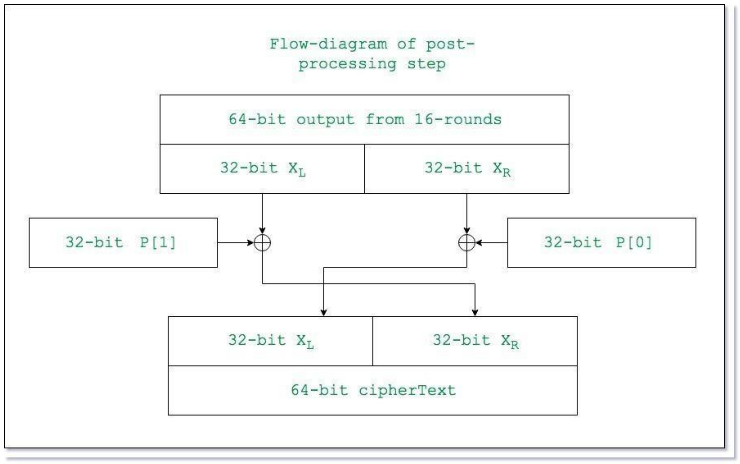


The description of the function ” F ” is as follows:

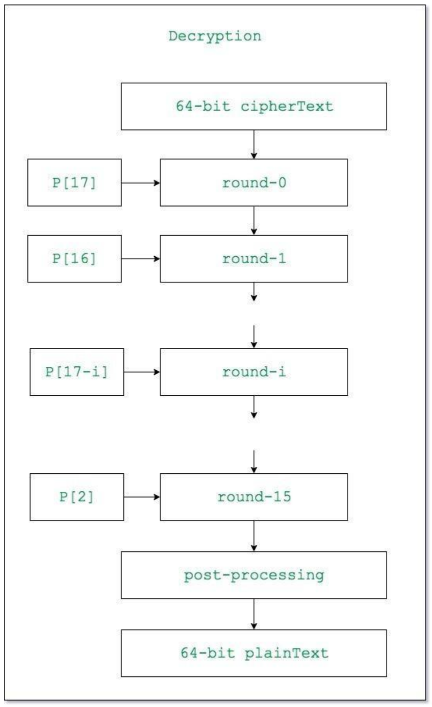


Here the function “add” is addition modulo 2^32.

2. Post-processing: The output after the 16 rounds is processed as follows:



Decryption



The decryption process is similar to that of encryption and the subkeys are used in reverse{P[17] – P[0]}. The entire decryption process can be elaborated as:

Lets see each step one by one:

Step1: Generation of subkeys:

- 18 subkeys{P[0]…P[17]} are needed in decryption process.
- These 18 subkeys are stored in a P-array with each array element being a 32-bit entry.
- It is initialised with the digits of pi.

The hexadecimal representation of each of the subkeys is given by:

P[0] = "243f6a88"

P[1] = "85a308d3"

.

P[17] = "8979fb1b"

- Note: See encryption for the initial values of P-array.

Now each of the subkeys is changed with respect to the input key as:

P[0] = P[0] xor 1st 32-bits of input key

P[1] = P[1] xor 2nd 32-bits of input key

.

P[i] = P[i] xor (i+1)th 32-bits of input key

(roll over to 1st 32-bits depending on the key length)

P[17] = P[17] xor 18th 32-bits of input key
(roll over to 1st 32-bits depending on key length)

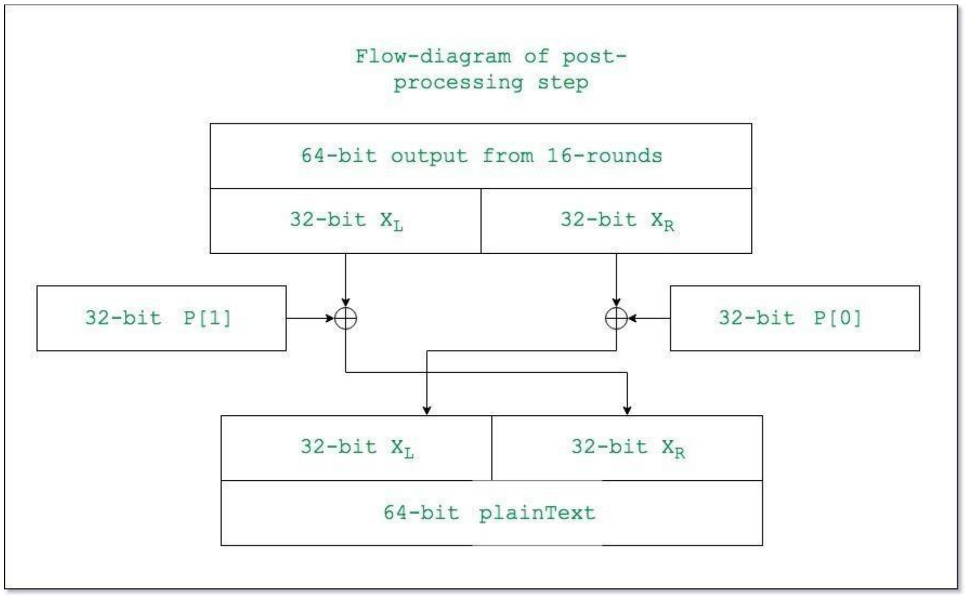
- The resultant P-array holds 18 subkeys that is used during the entire encryption process

Step2: initialise Substitution Boxes:

- 4 Substitution boxes(S-boxes) are needed{S[0]…S[4]} in both encryption as well as decryption process with each S-box having 256 entries{S[i][0]…S[i][255], 0&lei&le4} where each entry is 32-bit.
- It is initialised with the digits of pi(?) after initialising the P-array. You may find the s-boxes in here .

Step3: Decryption:

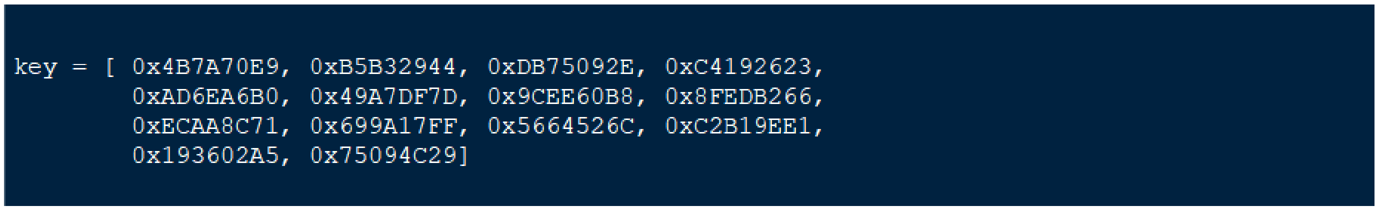
- The Decryption function also consists of two parts:
    1. Rounds: The decryption also consists of 16 rounds with each round(Ri)(as explained above) taking inputs the cipherText(C.T.) from previous round and corresponding subkey(P[17-i])(i.e for decryption the subkeys are used in reverse).
    2. Post-processing: The output after the 16 rounds is processed as follows:


Flow-diagram of post-processing step
64-bit output from 16-rounds
32-bit X_L
32-bit X_R
32-bit P[1]
32-bit P[0]
32-bit X_L
32-bit X_R
64-bit plainText

Chapter 3

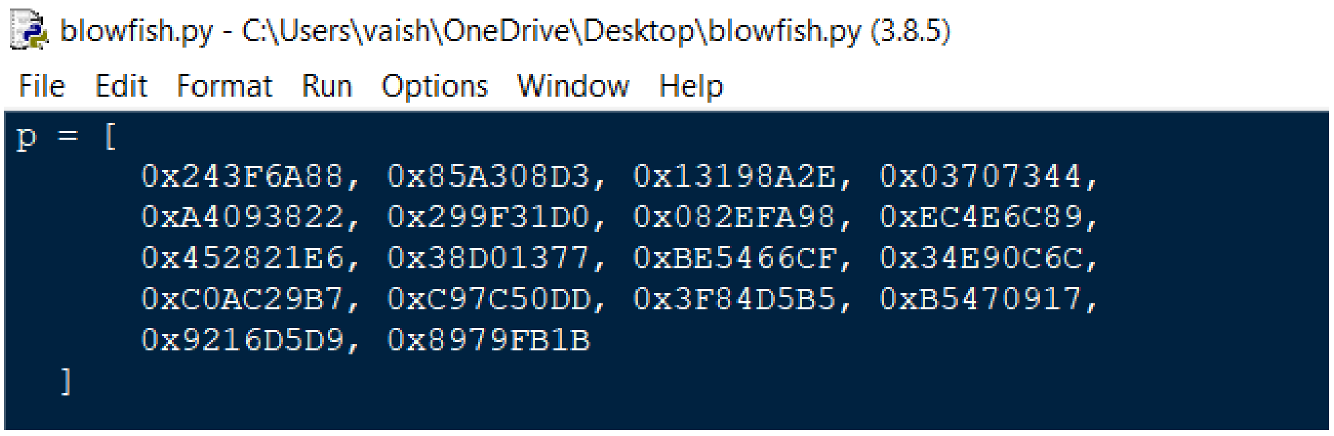
Results and Discussion

Fig1. Initializing the value for keys-



```
key = [ 0x4B7A70E9, 0xB5B32944, 0xDB75092E, 0xC4192623,
        0xAD6EA6B0, 0x49A7DF7D, 0x9CEE60B8, 0x8FEDB266,
        0xECAA8C71, 0x699A17FF, 0x5664526C, 0xC2B19EE1,
        0x193602A5, 0x75094C29]
```

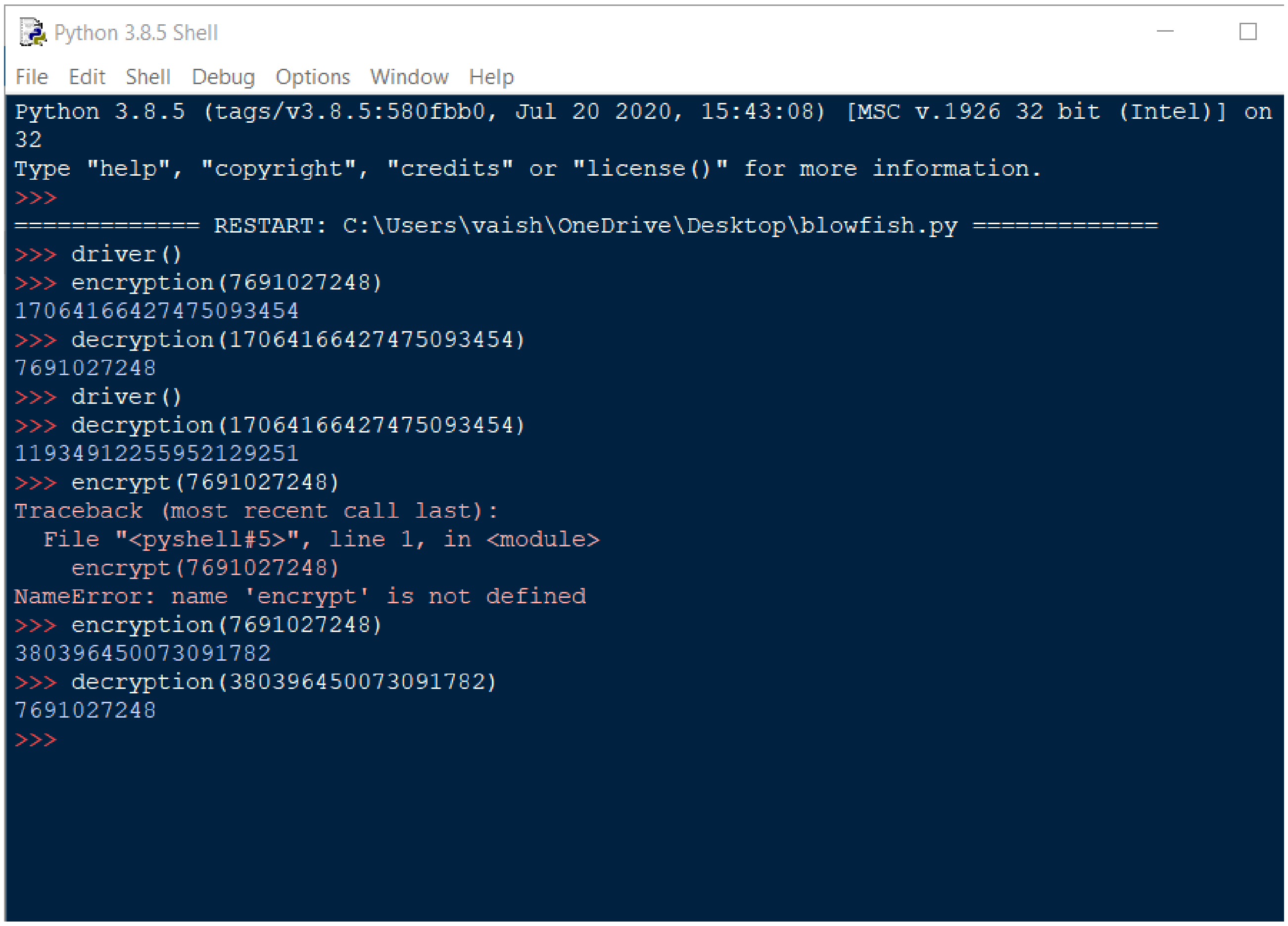
Fig2. Initializing the value for sub keys as the digits of pi-



blowfish.py - C:\Users\vaish\OneDrive\Desktop\blowfish.py (3.8.5)
File Edit Format Run Options Window Help

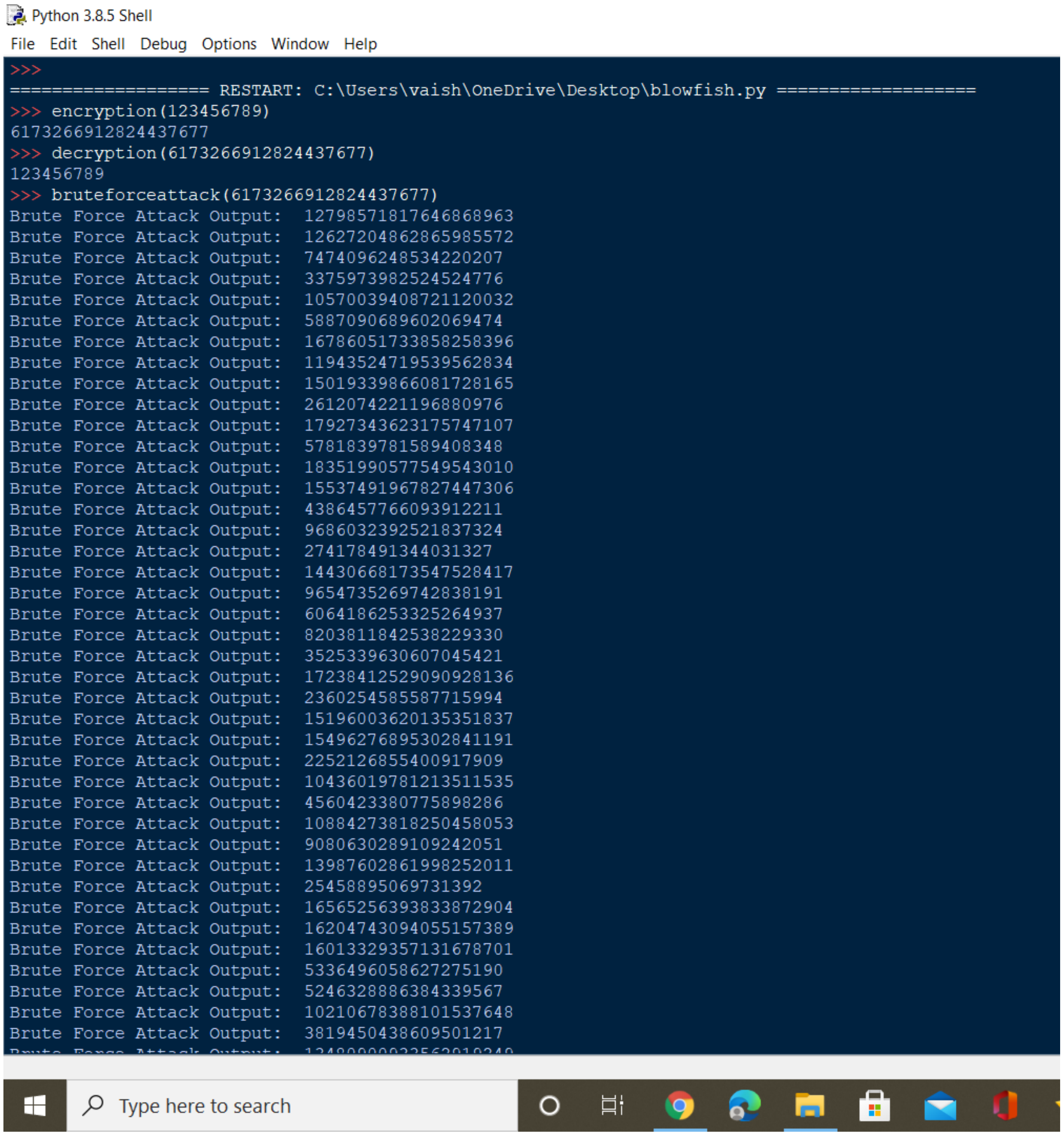
```
p = [
        0x243F6A88, 0x85A308D3, 0x13198A2E, 0x03707344,
        0xA4093822, 0x299F31D0, 0x082EFA98, 0xEC4E6C89,
        0x452821E6, 0x38D01377, 0xBE5466CF, 0x34E90C6C,
        0xC0AC29B7, 0xC97C50DD, 0x3F84D5B5, 0xB5470917,
        0x9216D5D9, 0x8979FB1B
   ]
```

Fig3. Encryption and decryption based on the value of keys stored in p array-



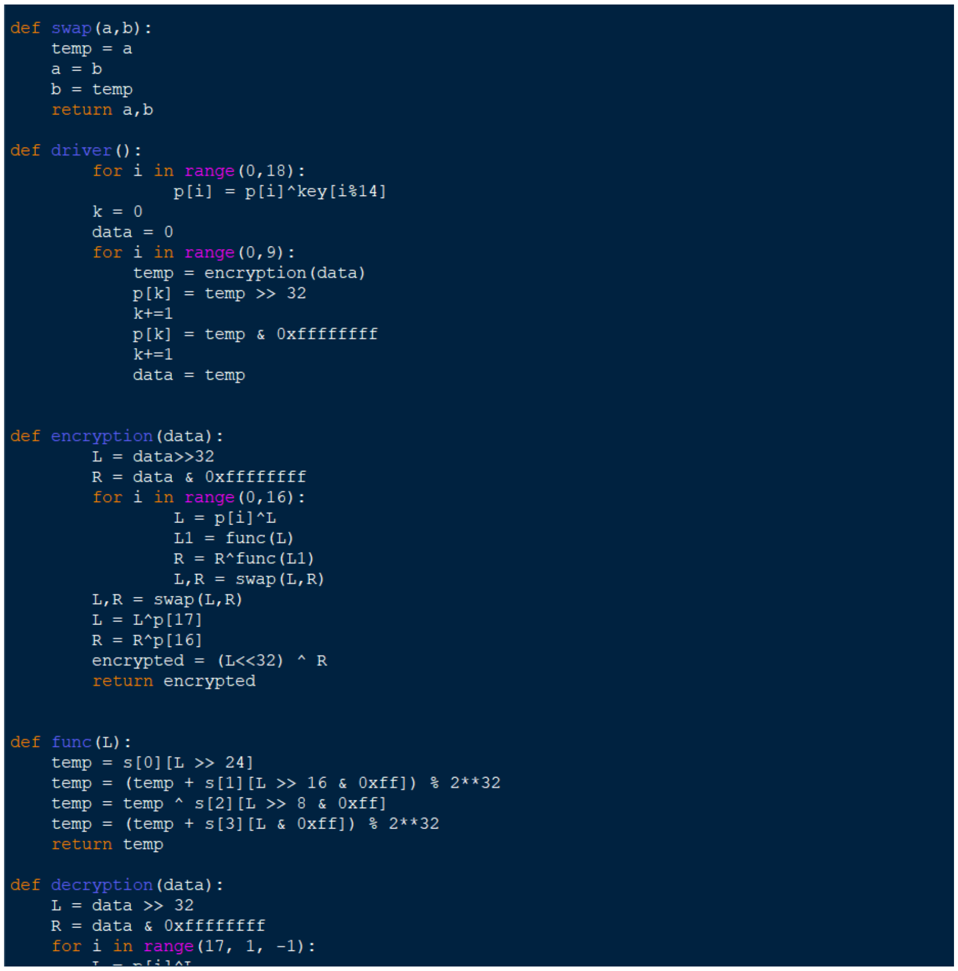
```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on
32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: C:\Users\vaish\OneDrive\Desktop\blowfish.py =============
>>> driver()
>>> encryption(7691027248)
17064166427475093454
>>> decryption(17064166427475093454)
7691027248
>>> driver()
>>> decryption(17064166427475093454)
11934912255952129251
>>> encrypt(7691027248)
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    encrypt(7691027248)
NameError: name 'encrypt' is not defined
>>> encryption(7691027248)
3803964500073091782
>>> decryption(3803964500073091782)
7691027248
>>>
```

Fig4. Brute Force Attack using the random keys for array P-



```
>>>
================== RESTART: C:\Users\vaish\OneDrive\Desktop\blowfish.py ==================
>>> encryption(123456789)
6173266912824437677
>>> decryption(6173266912824437677)
123456789
>>> bruteforceattack(6173266912824437677)
Brute Force Attack Output:  12798571817646868963
Brute Force Attack Output:  12627204862865985572
Brute Force Attack Output:  7474096248534220207
Brute Force Attack Output:  3375973982524524776
Brute Force Attack Output:  10570039408721120032
Brute Force Attack Output:  5887090689602069474
Brute Force Attack Output:  16786051733858258396
Brute Force Attack Output:  11943524719539562834
Brute Force Attack Output:  15019339866081728165
Brute Force Attack Output:  2612074221196880976
Brute Force Attack Output:  17927343623175747107
Brute Force Attack Output:  5781839781589408348
Brute Force Attack Output:  18351990577549543010
Brute Force Attack Output:  15537491967827447306
Brute Force Attack Output:  4386457766093912211
Brute Force Attack Output:  9686032392521837324
Brute Force Attack Output:  274178491344031327
Brute Force Attack Output:  14430668173547528417
Brute Force Attack Output:  9654735269742838191
Brute Force Attack Output:  6064186253325264937
Brute Force Attack Output:  8203811842538229330
Brute Force Attack Output:  3525339630607045421
Brute Force Attack Output:  17238412529090928136
Brute Force Attack Output:  2360254585587715994
Brute Force Attack Output:  15196003620135351837
Brute Force Attack Output:  15496276895302841191
Brute Force Attack Output:  2252126855400917909
Brute Force Attack Output:  10436019781213511535
Brute Force Attack Output:  4560423380775898286
Brute Force Attack Output:  10884273818250458053
Brute Force Attack Output:  9080630289109242051
Brute Force Attack Output:  13987602861998252011
Brute Force Attack Output:  254588950697313292
Brute Force Attack Output:  16565256393833872904
Brute Force Attack Output:  16204743094055157389
Brute Force Attack Output:  16013329357131678701
Brute Force Attack Output:  5336496058627275190
Brute Force Attack Output:  5246328886384339567
Brute Force Attack Output:  10210678388101537648
Brute Force Attack Output:  3819450438609501217
```

Fig5. Python Code for Encryption and Decryption



```python
def swap(a,b):
    temp = a
    a = b
    b = temp
    return a,b

def driver():
        for i in range(0,18):
                p[i] = p[i]^key[i%14]
        k = 0
        data = 0
        for i in range(0,9):
            temp = encryption(data)
            p[k] = temp >> 32
            k+=1
            p[k] = temp & 0xffffffff
            k+=1
            data = temp


def encryption(data):
        L = data>>32
        R = data & 0xffffffff
        for i in range(0,16):
                L = p[i]^L
                L1 = func(L)
                R = R^func(L1)
                L,R = swap(L,R)
        L,R = swap(L,R)
        L = L^p[17]
        R = R^p[16]
        encrypted = (L<<32) ^ R
        return encrypted


def func(L):
    temp = s[0][L >> 24]
    temp = (temp + s[1][L >> 16 & 0xff]) % 2**32
    temp = temp ^ s[2][L >> 8 & 0xff]
    temp = (temp + s[3][L & 0xff]) % 2**32
    return temp

def decryption(data):
    L = data >> 32
    R = data & 0xffffffff
    for i in range(17, 1, -1):
```

Chapter 4

Conclusion and Future Work

Blowfish is a fast block cipher except when changing keys. Each new key requires a pre-processing equivalent to 4KB of text. It is faster and much better than DES Encryption. It uses a 64-bit block size which makes it vulnerable to birthday attacks.

A reduced round variant of blowfish is known to be susceptible to known plaintext attacks(2nd order differential attacks – 4 rounds).

This advanced Blowfish Algorithm is more efficient in energy consumption and security to reduce the consumption of battery power devices. In the new proposed model of Blowfish by further increasing the key length, Blowfish will provide better results.

References:

1. Blowfish Cipher, http://en.wikipedia.org/wiki/Blowfish_(cipher), Date accessed: July 4, 2014

2.Blowfish products, https://www.schneier.com/blowfish-products.html, Date accessed: June 24, 2014.