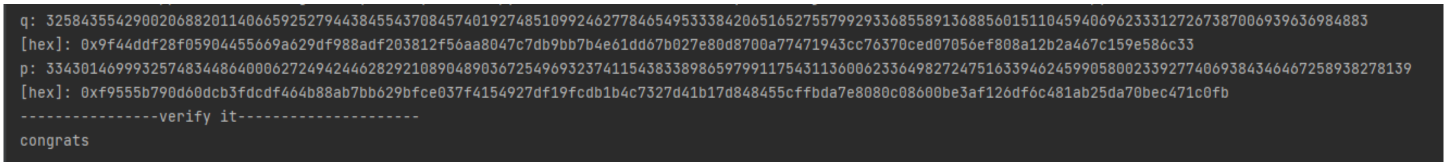
1.

(a) the RSA modulus N and both the correct result s and a faulty result s ´ of an RSA-CRT are used (Bellcore)

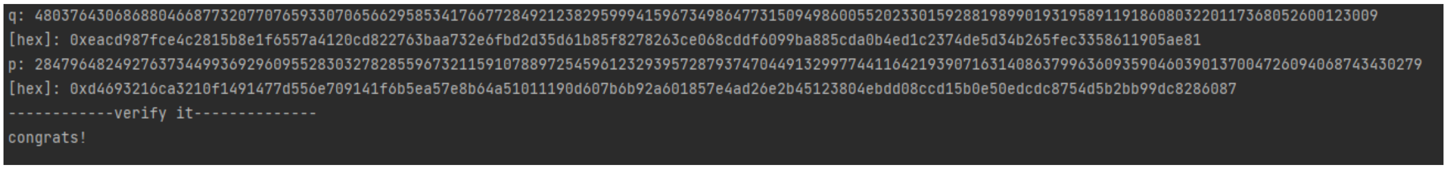
Ans: getting two integer and verify it by checking if n equals to q times p.
Code name: task1_a.py



```
q: 32584355429002068820114066592527944384554370845740192748510992462778465495333842065165275579929336855891368856015110459406962333127267387006939636984883
[hex]: 0x9f44ddf28f05904455669a629df988adf203812f56aa8047c7db9bb7b4e61dd67b027e80d8700a77471943cc76370ced07056ef808a12b2a467c159e586c33
p: 33430146999325748344864000627249424462829210890489036725496932374115438338986597991175431136006233649827247516339462459905800233927740693843464672589382781 39
[hex]: 0xf9555b790d60dcb3fdcdf464b88ab7bb629bfce037f4154927df19fcdb1b4c7327d41b17d848455cffbda7e8080c08600be3af126df6c481ab25da70bec471c0fb
----------------verify it---------------------
congrats
```

(b) the RSA modulus N, the public exponent e, the faulty result s ´ , and the message m are used (Lenstra)

Ans: getting two integer and verify by checking if n equals to q times p.
Code name: task1_b.py



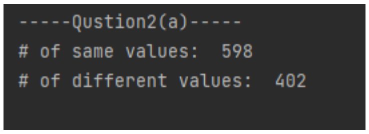
```
q: 48037643068688046687732077076593307065662958534176677284921238295999415967349864773150949860055202330159288198990193195891191860803220117368052600123009
[hex]: 0xeacd987fce4c2815b8e1f6557a4120cd822763baa732e6fbd2d35d61b85f8278263ce068cddf6099ba885cda0b4ed1c2374de5d34b265fec3358611905ae81
p: 2847964824927637344993692960955283032782855967321159107889725459612329395728793747044913299774411642193907163140863799636093590460390137004726094068743430279
[hex]: 0xd4693216ca3210f1491477d556e709141f6b5ea57e8b64a51011190d607b6b92a601857e4ad26e2b45123804ebdd08ccd15b0e50edcdc8754d5b2bb99dc8286087
------------verify it--------------
congrats!
```

2.

(a) Having a reliable fault injection setup is critical to successfully executing DFA. From the glitch data provided, how many glitches are successful? Does that seem like a high success rate?

Ans: In the total 1000 pairs of data, I found that there are 402 pairs of glitch data. And the remaining 598 are identical. This is a high success rate because we will not use all the provided data. What we really need is 8 pair of different glitch data that can be used to attack on the four columns.

Code name: task2_a_b.py

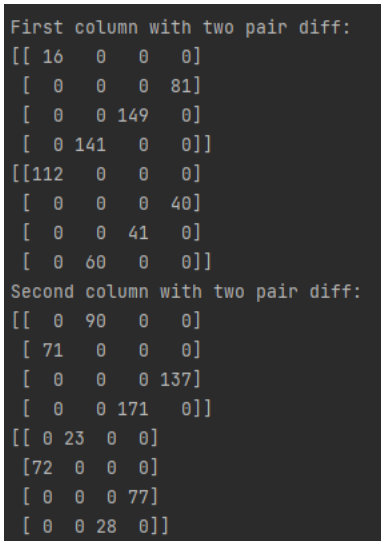


```
-----Qustion2(a)-----
# of same values:  598
# of different values:  402
```

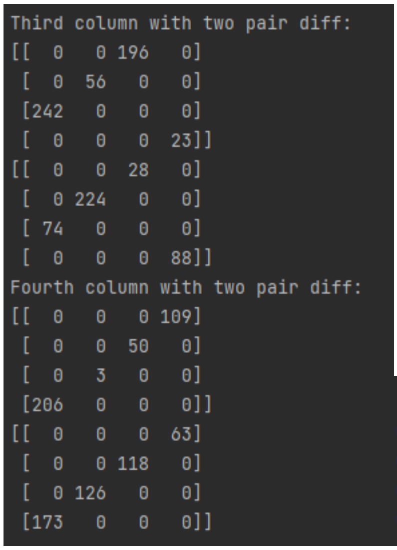
(b) To perform the DFA attack described by Piret and Quisquater, we will need to find multiple ciphertext/faultytext pairs. How many total pairs will we need? By parsing the provided files, find enough pairs to complete the attack and output them here in hex format. Can anything happen that might cause you to need more pairs?

Ans: We need "8 pairs" to perform the full DFA attack. In this homework, we only need 8 pairs to do the full DFA attack. However, I believe in the real world, we may have to prepare more pairs if two pairs is not enough to find out the corresponding keybytes. For example, the final filtering for loop can not find the match keybytes are return a null back. In this situation, I think we have to prepare more pair to do the full DFA attack.

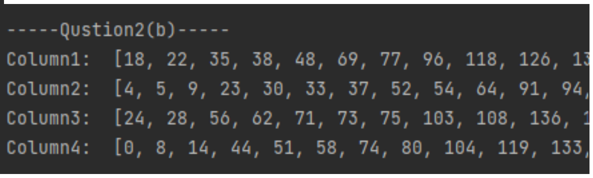
Code name: task2_a_b.py



```
First column with two pair diff:
[[ 16   0   0   0]
 [  0   0   0  81]
 [  0   0 149   0]
 [  0 141   0   0]]
[[112   0   0   0]
 [  0   0   0  40]
 [  0   0  41   0]
 [  0  60   0   0]]
Second column with two pair diff:
[[  0  90   0   0]
 [ 71   0   0   0]
 [  0   0   0 137]
 [  0   0 171   0]]
[[ 0 23  0  0]
 [72  0  0  0]
 [ 0  0  0 77]
 [ 0  0 28  0]]
```



```
Third column with two pair diff:
[[  0   0 196   0]
 [  0  56   0   0]
 [242   0   0   0]
 [  0   0   0  23]]
[[  0   0  28   0]
 [  0 224   0   0]
 [ 74   0   0   0]
 [  0   0   0  88]]
Fourth column with two pair diff:
[[  0   0   0 109]
 [  0   0  50   0]
 [  0   3   0   0]
 [206   0   0   0]]
[[  0   0   0  63]
 [  0   0 118   0]
 [  0 126   0   0]
 [173   0   0   0]]
```

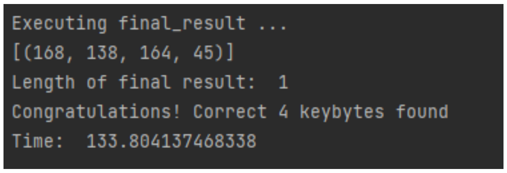


```
-----Qustion2(b)-----
Column1:  [18, 22, 35, 38, 48, 69, 77, 96, 118, 126, 13
Column2:  [4, 5, 9, 23, 30, 33, 37, 52, 54, 64, 91, 94,
Column3:  [24, 28, 56, 62, 71, 73, 75, 103, 108, 136, 1
Column4:  [0, 8, 14, 44, 51, 58, 74, 80, 104, 119, 133,
```

The figure shows “which row of data” that can be used to do the Full DFA attack. And also prove it by using “XOR” to double if that is suitable for each column attack.

(c) Simple Piret and Quisquater DFA: We will provide pairs with glitch in the first byte. Recover 4 bytes of the key.

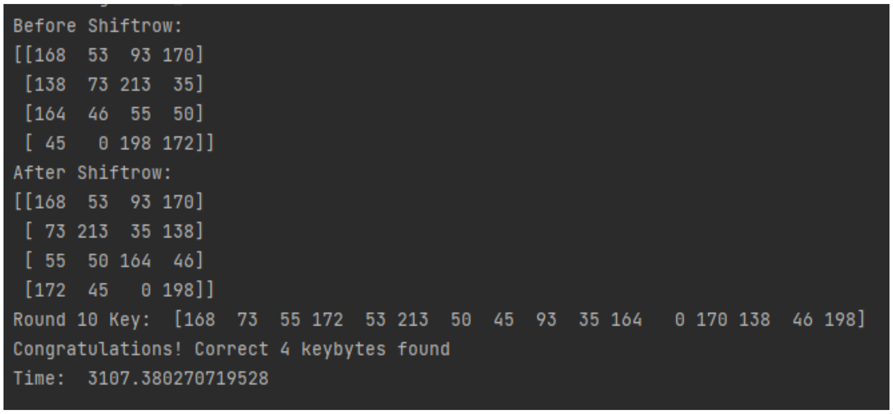
Key: (168, 138, 164, 45)
Code name: task2_c.py



```
Executing final_result ...
[(168, 138, 164, 45)]
Length of final result:  1
Congratulations! Correct 4 keybytes found
Time:  133.804137468338
```

(d) Full Piret and Quisquater DFA: using the provided encryption/glitch data, recover the entire round 10 key.

Round 10 Key: [168 73 55 172 53 213 50 45 93 35 164   0 170 138 46 198]
Time: about 50 minutes
Code name : task2_d.py

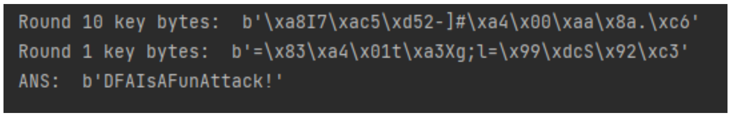


```
Before Shiftrow:
[[168  53  93 170]
 [138  73 213  35]
 [164  46  55  50]
 [ 45   0 198 172]]
After Shiftrow:
[[168  53  93 170]
 [ 73 213  35 138]
 [ 55  50 164  46]
 [172  45   0 198]]
Round 10 Key:  [168  73  55 172  53 213  50  45  93  35 164   0 170 138  46 198]
Congratulations! Correct 4 keybytes found
Time:  3107.380270719528
```

(e) Recover the original (first round) key and use it to decrypt the following (hex encoded) secret message:
2a92fc6ad8006b658f49062c2843ad99

Ans: Successfully get the decoded message: b'DFAIsAFunAttack!', that's really cool.

Code name: task2_e.py



```
Round 10 key bytes:  b'\xa8I7\xac5\xd52-]#\xa4\x00\xaa\x8a.\xc6'
Round 1 key bytes:  b'=\x83\xa4\x01t\xa3Xg;l=\x99\xdcS\x92\xc3'
ANS:  b'DFAIsAFunAttack!'
```