ASSIGNMENT 3
CS666: HARDWARE SECURITY OF INTERNET OF THINGS

Group G2 Oct, 2023

1. Group Members

1. VIKRANT CHAUHAN : MS CYBERSERCURITY : 231110407
2. SOUVIK MUKHERJEE : MS CYBERSECURITY : 231110405
3. VISHAL KUMAR : MTECH CYBERSECURITY : 231110058
4. VALETI LOKESH : MS CYBERSECURITY : 231110406
5. M DHILIPKUMAR : MTECH CYBERSECURITY : 231110026

2. Helping instruction for running the code files

- Kindly do *cd* to the specific folder
- Run the command python *Q1_Group2.py*
- See Output

IMPORTANT NOTE:- Question 1 Program Execution takes about 5 to 10 minutes for running its our humblest request to please keep patience

3. Question

Question 1:

You will be provided with the power traces of AES. The power traces are stored in a CSV file, where each row indicates the power consumption of one AES execution. For every row, the first entry is plaintext, the second entry is ciphertext, and all the subsequent entries are power consumption values. Your task is to write a code for Difference of Mean Attack, and use that code on the given power traces to recover the 4th and 5th bytes of the secret key used in the AES execution. You will be provided with

5 CSV files, you have to use one according to your group number. (e.g., if your group number is 1, then use HW power trace 1.csv)

Introduction :

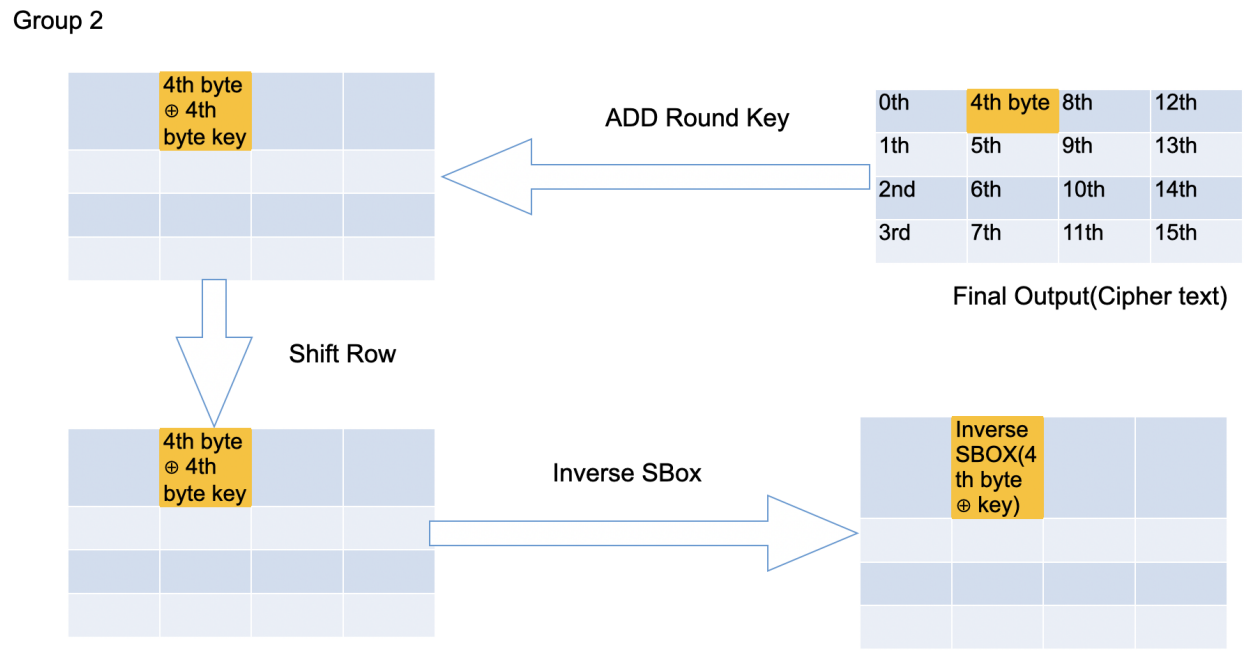


Figure 1: 4th Byte Analysis

From the above figure we can understand how we can actually make use of cipher text to recover the 4th byte of the 10th round key. Actually what we do is that on the cipher text we we xor the guessed 4th byte of the key.After that we apply shift rows and then we apply inverse sbox operation.After that the result which we get we put that into zero bin or one bin based on the LSB of the received output.If the LSB is one then we put the power trace in one bin otherwise if our LSB is zero then we put the power trace in zero bin.

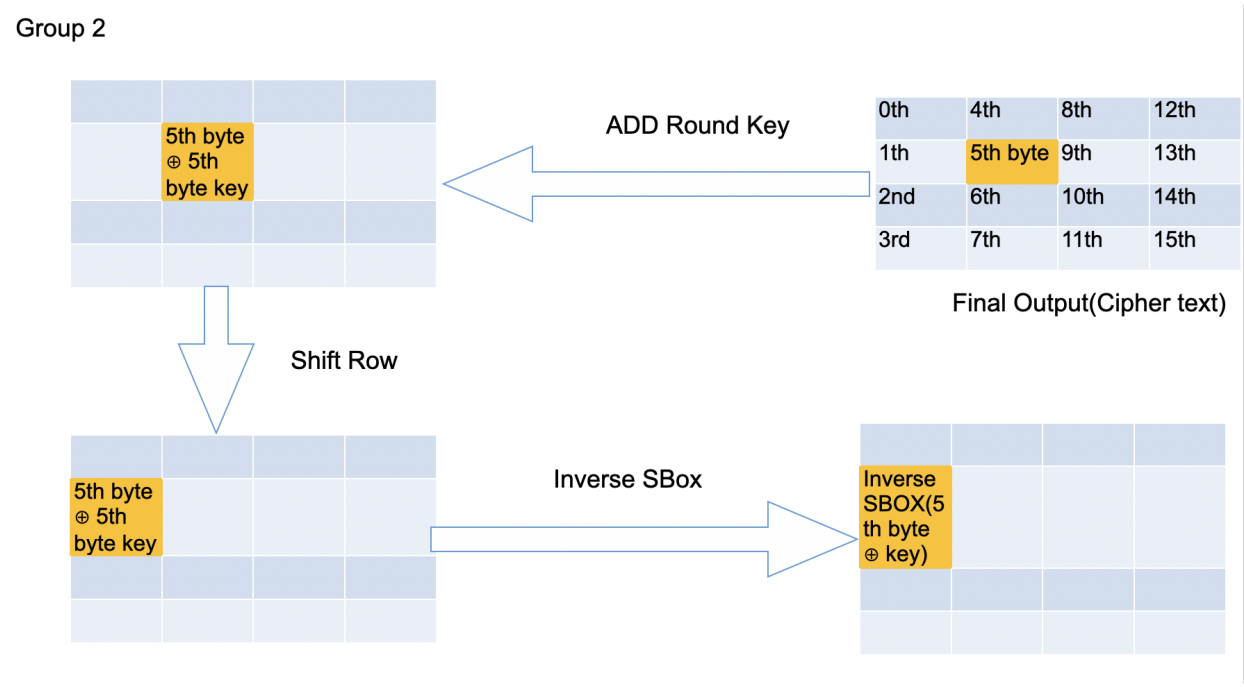


Figure 2: 5th Byte Analysis

From the above figure we can understand how we can actually make use of cipher text to recover the 5th byte of the 10th round key. Actually what we do is that on the cipher text we we xor the guessed 5th byte of the key.After that we apply shift rows and then we apply inverse sbox operation.After that the result which we get we put that into zero bin or one bin based on the LSB of the received output.If the LSB is one then we put the power trace in one bin otherwise if our LSB is zero then we put the power trace in zero bin.

Algorithm For Finding The 4th Byte of the 10th Round Key:

We are given power traces along with the corresponding ciphertext and plaintext pairs.Our aim is to with the help of these pairs we have to extract the 4th and the 5th byte of the 10th round key. Our algorithm is as follows:-

1. Open the `HW_power_trace_2.csv` file then extract from it the power traces and ciphertext.Store the power trace values in a 2 dimensional array named complete_power_array and also store the cipher texts in ciphertext array.

2. Now we make an array of guessed 4th byte keys.We know that a byte of key is of 8 bits therefore the value of the key can be from 0 to 255 only. We store all these 0 to 255 values in a keys array.

3. Now for each ciphertext we extract the 4th byte of the cipher text and then perform xor with the guessed key and then take inverse sbox operation.

4. After this see if the LSB is 1 then put the power trace in one bin otherwise if LSB is 0 then put the power trace in 0th bin.

5. Now do this 256 times for all 10000 ciphertexts, this is because we have 10000 ciphertexts and for each ciphertext we have 256 key guesses.

6. After this for each key guess value we will find the difference of mean with the help of zero and one bin arrays. Also we store these difference of mean value values in a difference of mean array.

7. After this we pass the difference of mean array to a ranking function which returns a rank array which gives rank of the difference of mean greater is the dom value greater is the rank.

8. We repeat steps 3 to 7 steps for 11 times this is because for each ciphertext we have 11 corresponding power trace values this is why we do these same steps 11 times.This is because we have 10 AES rounds + 1 initial add round key rounds this is the reason why we are getting 11 power traces.

9. Now after this we sum up the rank value for each of the key value which is stored in the ranking array.We store this value in total rank array.

10. Now in the total rank array we see which key has the highest value.The key with the highest value is our 4th byte of the key.In this way we get the 4th byte of the array.

FlowChart:

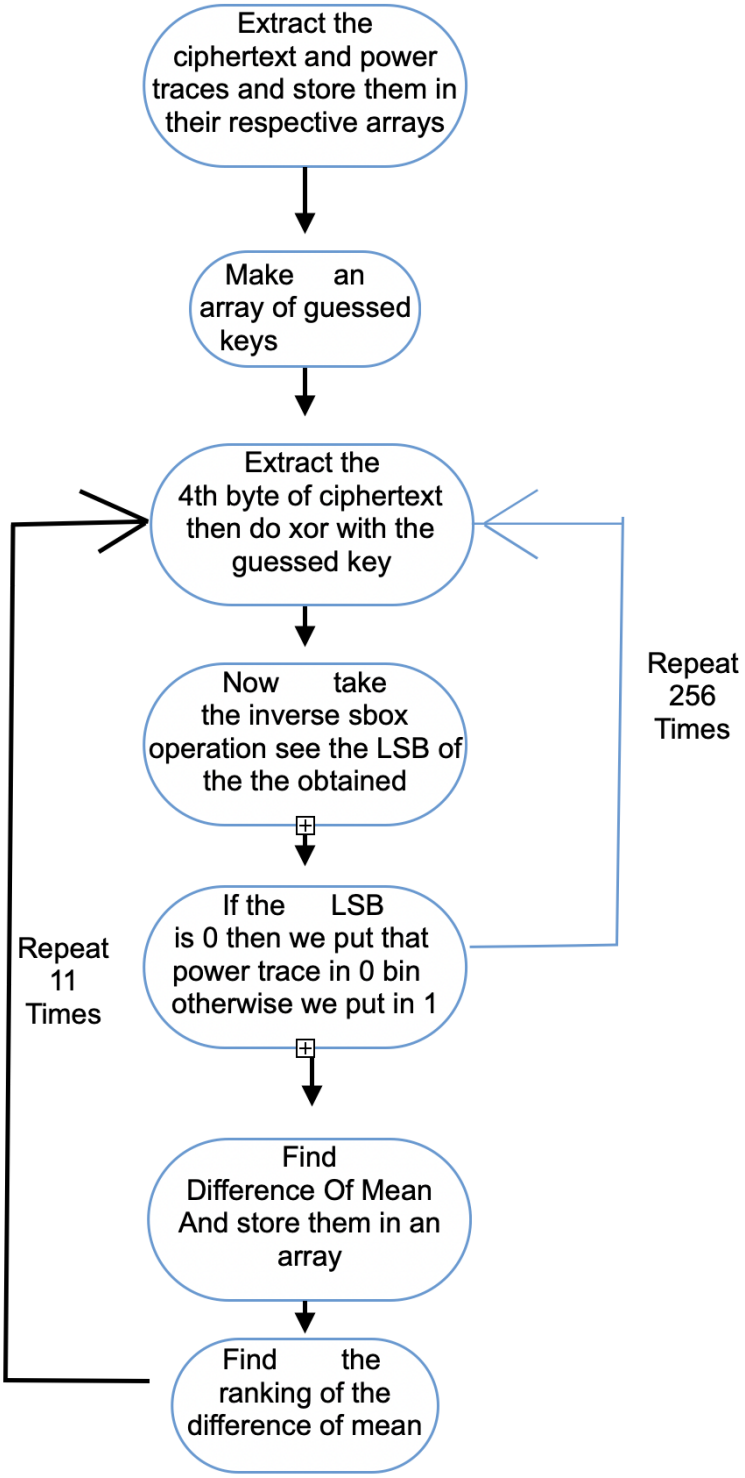


Figure 3: Flowchart For 4th Byte Analysis

Algorithm For Finding The 5th Byte of the 10th Round Key:

We are given power traces along with the corresponding ciphertext and plaintext pairs.Our aim is to with the help of these pairs we have to extract the 5th byte of the 10th round key. Our algorithm is as follows:-

1. Open the `HW_power_trace_2.csv` file then extract from it the power traces and ciphertext.Store the power trace values in a 2 dimensional array named complete_power_array and also store the cipher texts in ciphertext array.

2. Now we make an array of guessed 4th byte keys.We know that a byte of key is of 8 bits therefore the value of the key can be from 0 to 255 only. We store all these 0 to 255 values in a keys array.

3. Now for each ciphertext we extract the 5th byte of the cipher text and then perform xor with the guessed key and then take inverse sbox operation.

4. After this see if the LSB is 1 then put the power trace in one bin otherwise if LSB is 0 then put the power trace in 0th bin.

5. Now do this 256 times for all 10000 ciphertexts, this is because we have 10000 ciphertexts and for each ciphertext we have 256 key guesses.

6. After this for each key guess value we will find the difference of mean with the help of zero and one bin arrays. Also we store these difference of mean value values in a difference of mean array.

7. After this we pass the difference of mean array to a ranking function which returns a rank array which gives rank of the difference of mean greater is the dom value greater is the rank.

8. We repeat steps 3 to 7 steps for 11 times this is because for each ciphertext we have 11 corresponding power trace values this is why we do these same steps 11 times.This is because we have 10 AES rounds + 1 initial add round key rounds this is the reason why we are getting 11 power traces.

9. Now after this we sum up the rank value for each of the key value which is stored in the ranking array.We store this value in total rank array.

10. Now in the total rank array we see which key has the highest value.The key with the highest value is our 5th byte of the key.In this way we get the 5th byte of the array.

Flowchart:

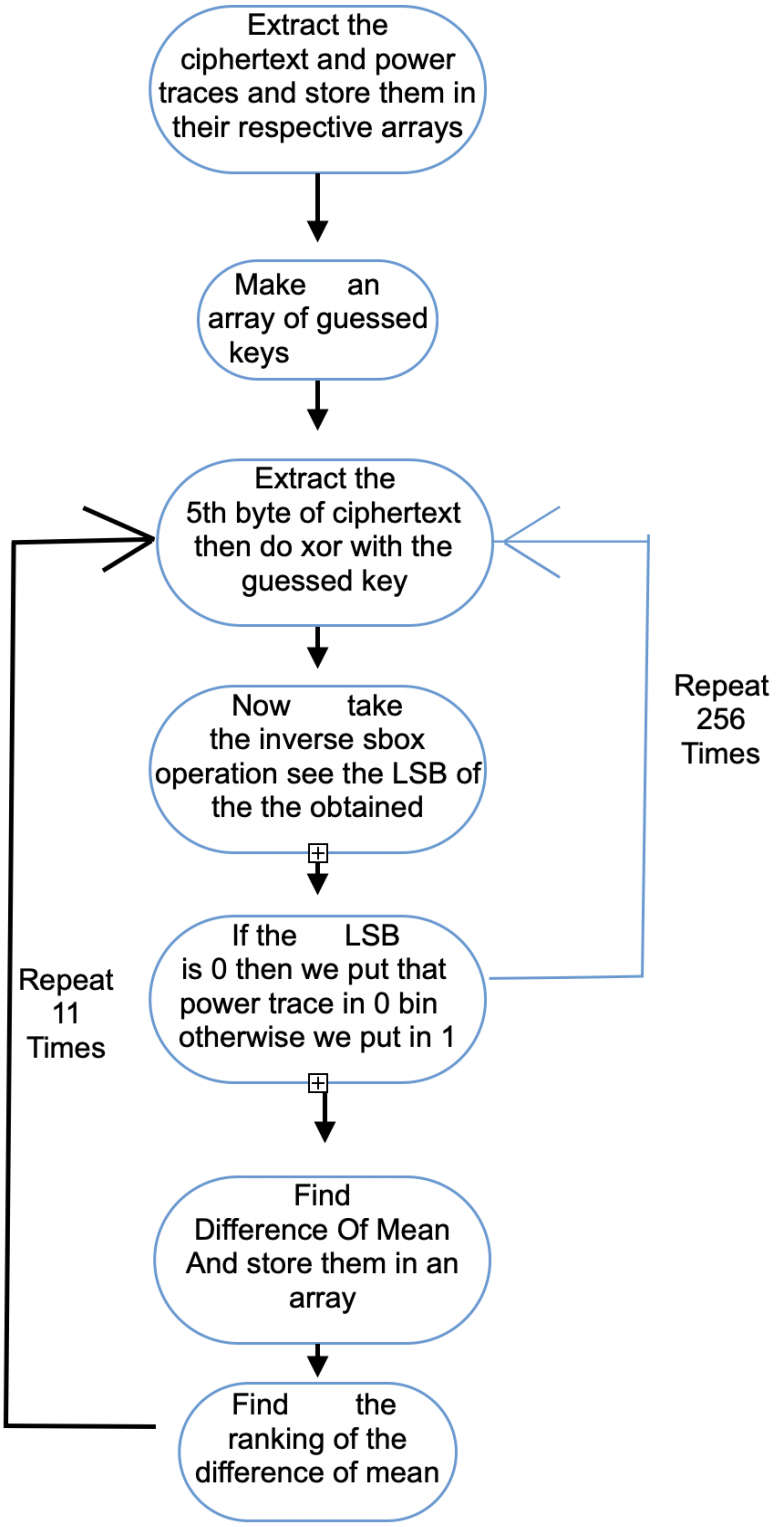
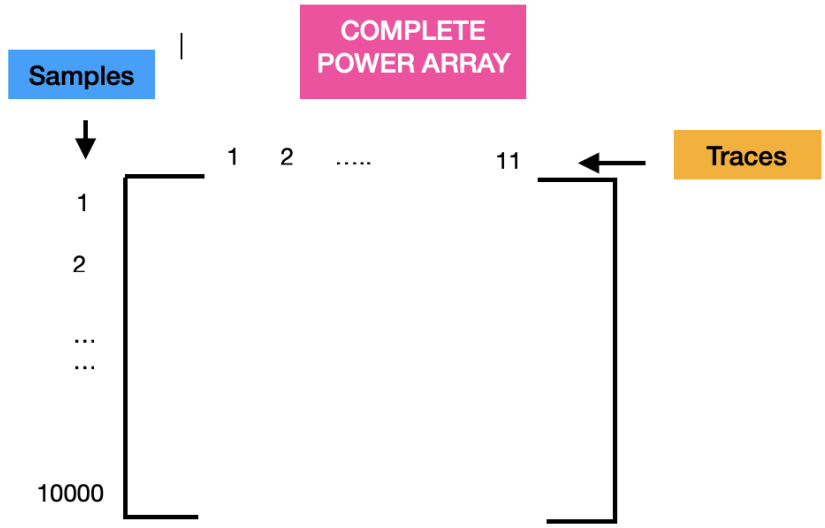
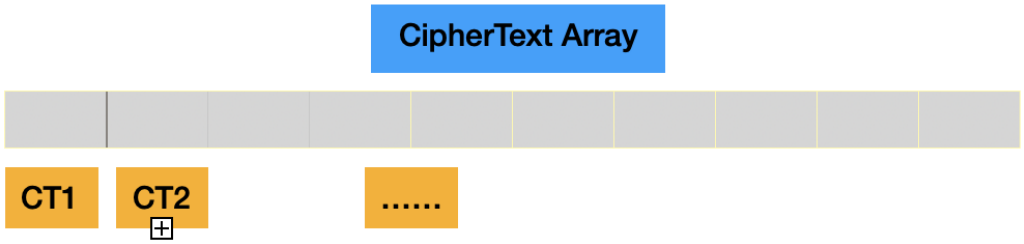
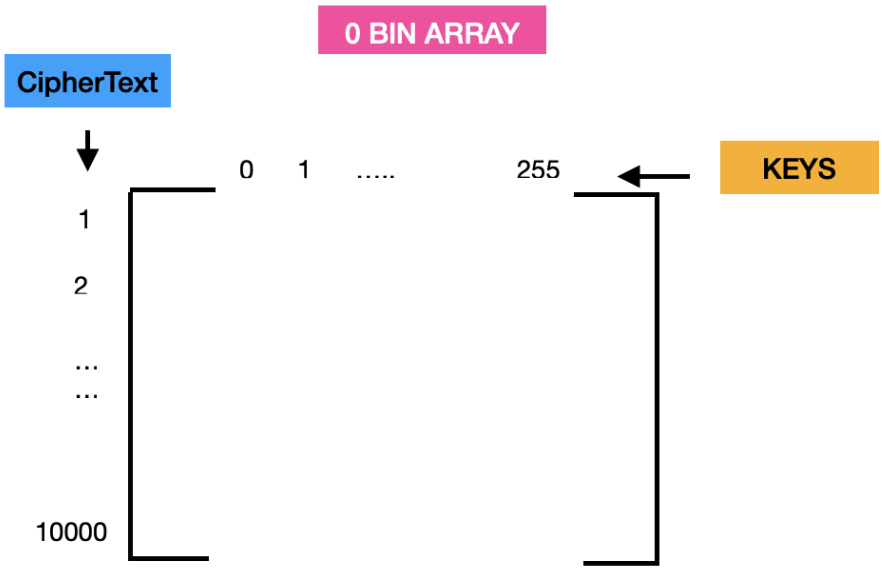


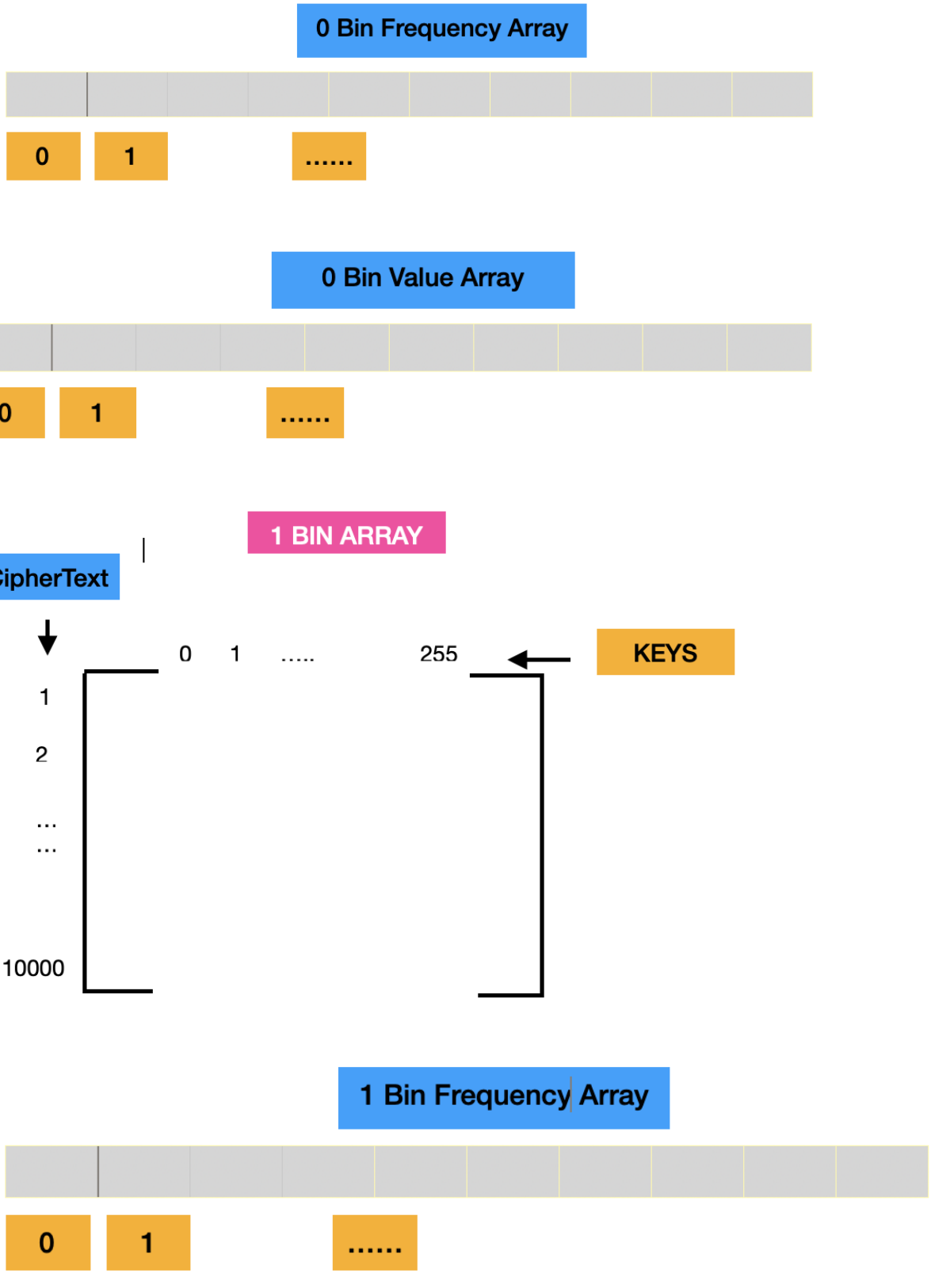
Figure 4: Flowchart For 5th Byte Analysis

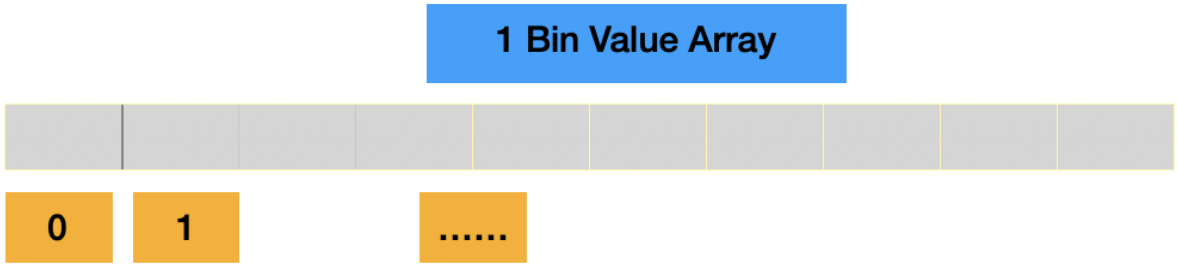
Matrices Used:

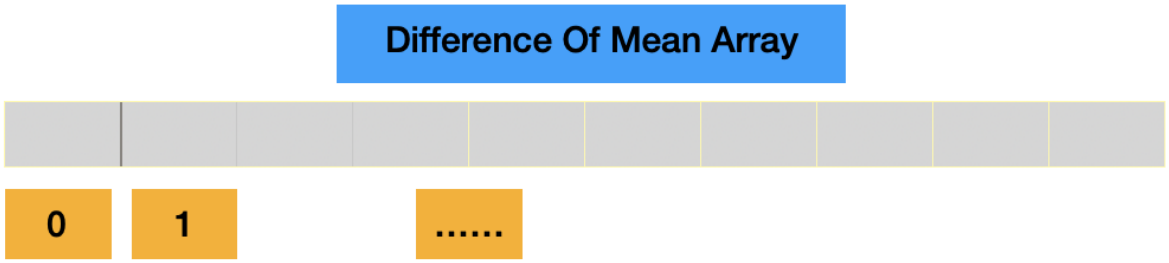


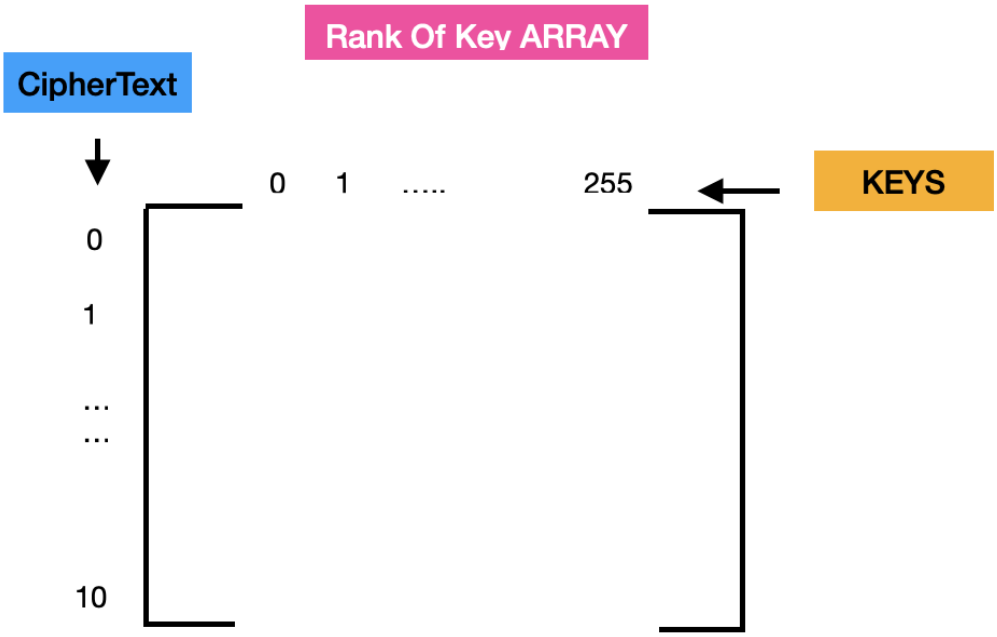


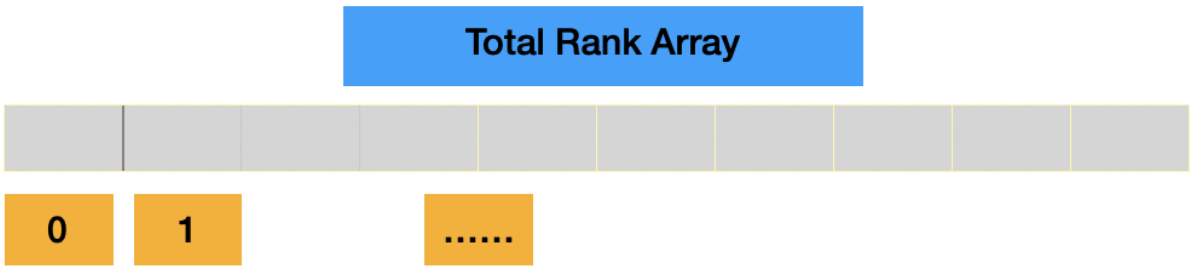



0 Bin Frequency Array
0
1
.....
0 Bin Value Array
0
1
.....
1 BIN ARRAY
CipherText
0
1
.....
255
KEYS
1
2
...
...
10000
1 Bin Frequency Array
0
1
.....


1 Bin Value Array
0
1
.....


Difference Of Mean Array
0
1
.....


Rank Of Key ARRAY
CipherText
0
1
.....
255
KEYS
0
1
...
...
10



OUTPUT For Finding 4th Byte Of Key:-

```
vikrantchauhan@Vikrants-MacBook-Air Assignmet 3 % python3 Q1_Group2.py
**********************************************************************
Q1 Group 2 Solution
**********************************************************************
List Of Guessed Keys
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45
63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104,
119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153
, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 2
16, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250,
**********************************************************************
Starting iteration for 1 th Power Trace
Zero Bin Value  [[63, 63, 0, 63, 0, 0, 63, 63, 0, 0, 0, 0, 0, 0, 0, 63, 0, 63, 63, 63, 0, 63, 63, 63, 63, 63, 63, 0, 63, 63, 0, 63, 63, 63, 0, 63, 63, 0, 63, 0, 0, 0, 63, 63
 0, 0, 0, 0, 0, 63, 63, 0, 63, 63, 63, 0, 63, 63, 0, 0, 0, 0, 0, 63, 0, 0, 0, 63, 0, 63, 63, 63, 0, 0, 63, 63, 63, 0, 0, 63, 63, 0, 63, 0, 63, 0, 0, 0, 0, 0, 63, 0, 0, 0,
0, 63, 63, 0, 0, 0, 63, 63, 63, 0, 0, 0, 0, 0, 63, 0, 63, 0, 63, 63, 63, 63, 63, 63, 0, 0, 0, 0, 0, 63, 0, 63, 0, 63, 63, 63, 0, 0, 63, 0, 63, 63, 63, 0, 0, 63, 63, 63, 0, 0
, 63, 63, 0, 0, 0, 0, 63, 0, 0, 0, 0, 0, 63, 63, 0, 63, 63, 0, 63, 0, 0, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 0, 0, 63, 0, 0, 63, 0, 63, 0, 0, 63, 0, 63,
 0, 71, 0, 0, 0, 0, 0, 0, 0, 71, 71, 0, 0, 71, 0, 71, 71, 71, 71, 71, 71, 0, 0, 0, 0, 0, 71, 71, 0, 0, 71, 71, 0, 0, 0, 71, 0, 71, 71, 0, 0, 0, 71, 0, 71, 71, 0, 71, 71, 71,
1, 71, 71, 0, 71, 71, 0, 0, 0, 0, 0, 0, 0, 71, 0, 71, 0, 0, 0, 71, 0, 0, 0, 71, 0, 0, 0, 71, 0, 0, 0, 0, 0, 0, 71, 0, 71, 0, 0, 71, 71, 0, 0, 71, 0, 71, 71, 71, 71, 71, 71, 0,
, 71, 71, 71, 0, 71, 71, 0, 0, 0, 0, 0, 0, 71, 71, 71, 0, 0, 71, 71, 71, 0, 71, 0, 0, 71, 71, 71, 0, 71, 0, 71, 0, 0, 0, 0, 71, 0, 71, 71, 0, 71, 71, 0, 0, 0, 0, 0, 71, 0, 0
 71, 71, 0, 71, 0, 0, 71, 0, 71, 0, 0, 71, 0, 0, 71, 71, 71, 71, 71, 71, 71, 71, 71, 71, 71, 71, 71, 71, 0, 0], [0, 0, 0, 0, 0, 59, 0, 59, 0, 59, 59, 59, 59, 59, 59, 0, 59,
, 59, 59, 0, 0, 0, 0, 0, 0, 0, 59, 59, 0, 0, 0, 0, 0, 59, 0, 59, 0, 59, 59, 59, 0, 0, 59, 0, 59, 0, 0, 0, 59, 0, 0, 0, 0, 0, 59, 59, 0, 59, 59, 0, 59, 0, 59, 59, 0, 59, 0, 0, 5
0, 59, 59, 59, 59, 0, 0, 59, 59, 59, 59, 59, 59, 59, 59, 59, 59, 59, 59, 59, 59, 0, 59, 59, 59, 0, 59, 59, 59, 59, 59, 59, 0, 59, 59, 0, 59, 59, 59, 0, 59, 0, 0, 59, 59, 0,
59, 59, 59, 59, 0, 59, 59, 0, 59, 0, 0, 0, 59, 59, 0, 59, 0, 0, 0, 0, 0, 59, 0, 0, 0, 59, 0, 59, 59, 59, 0, 0, 59, 59, 0, 0, 0, 0, 0, 59, 59, 0, 59, 59, 59, 0, 59, 59, 0, 0,
 0, 0, 59, 0, 59, 0, 0, 0, 0, 0, 59], [0, 0, 0, 0, 72, 72, 72, 72, 72, 0, 0, 72, 72, 0, 0, 72, 72, 72, 0, 0, 0, 0, 0, 72, 0, 72, 72, 72, 72, 0, 72, 0, 72, 0, 72, 72, 0, 72,
```

Figure 5: 4th Byte analysis with 1st Power Trace

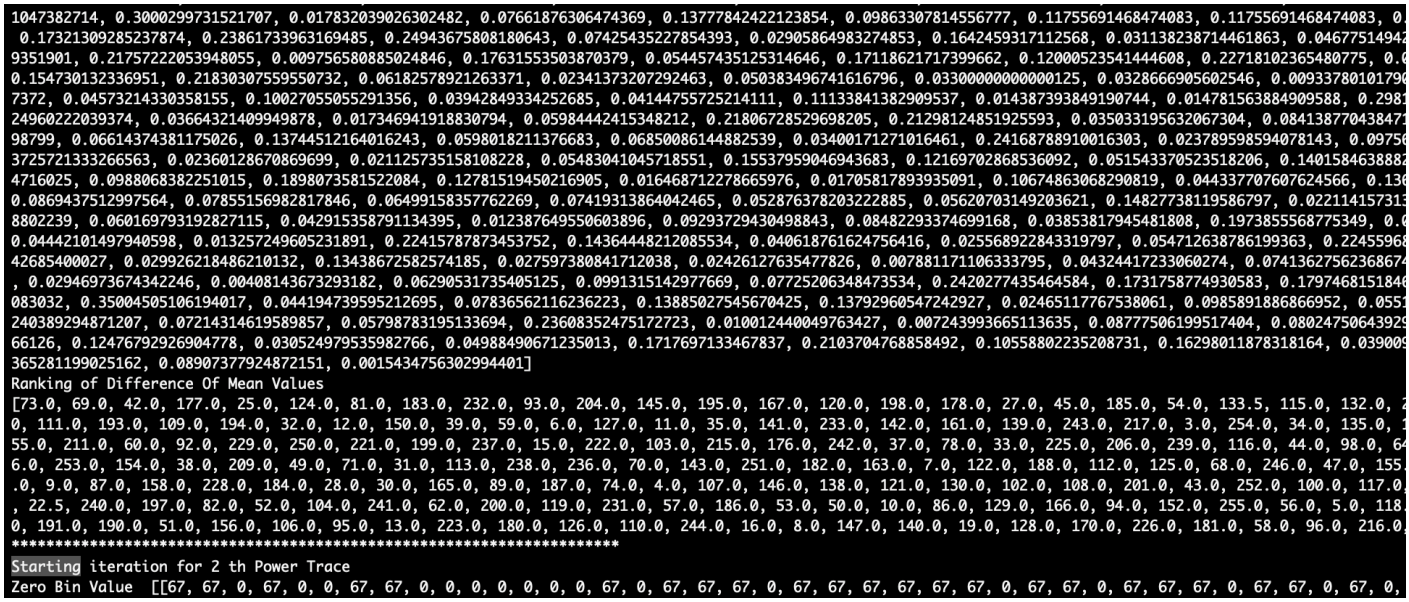


Figure 6: 4th Byte analysis with 2nd Power Trace

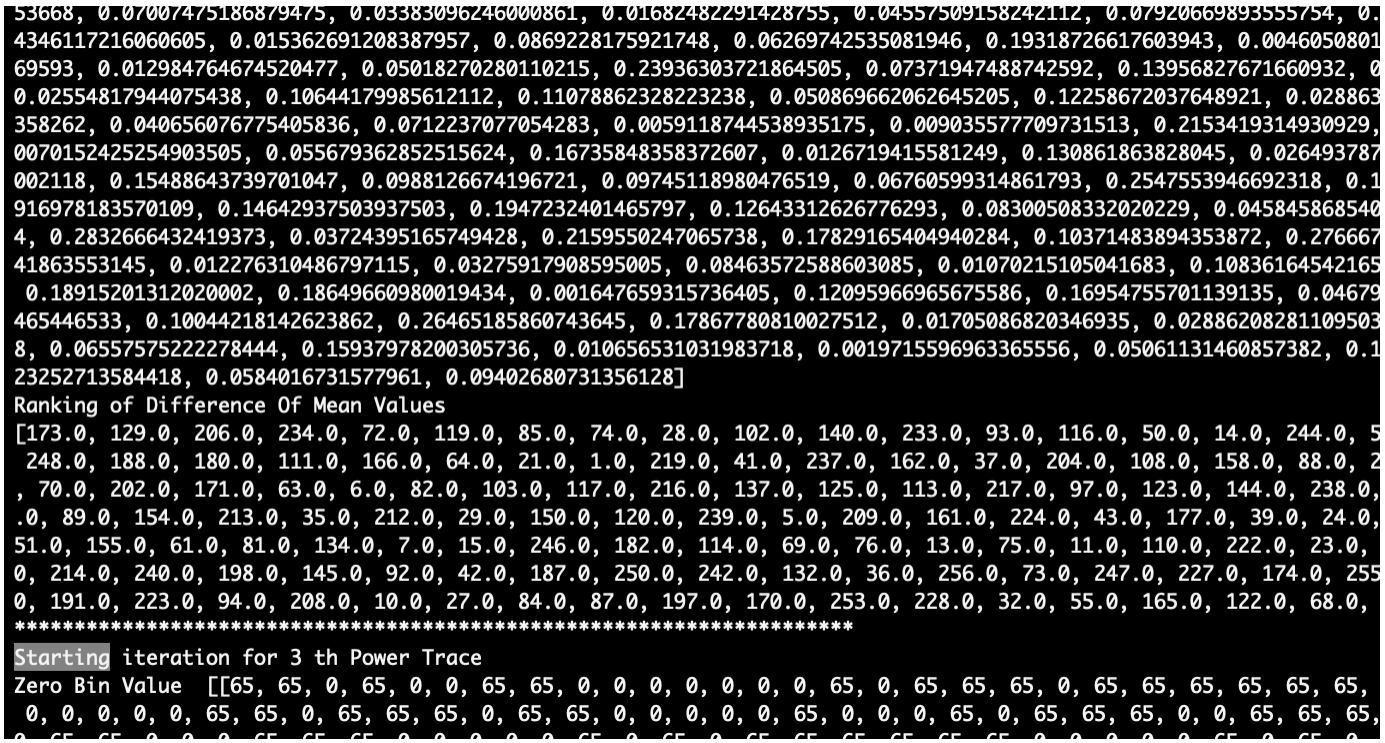


Figure 7: 4th Byte analysis with 3rd Power Trace

Figure 8: 4th Byte analysis with 11th Power Trace

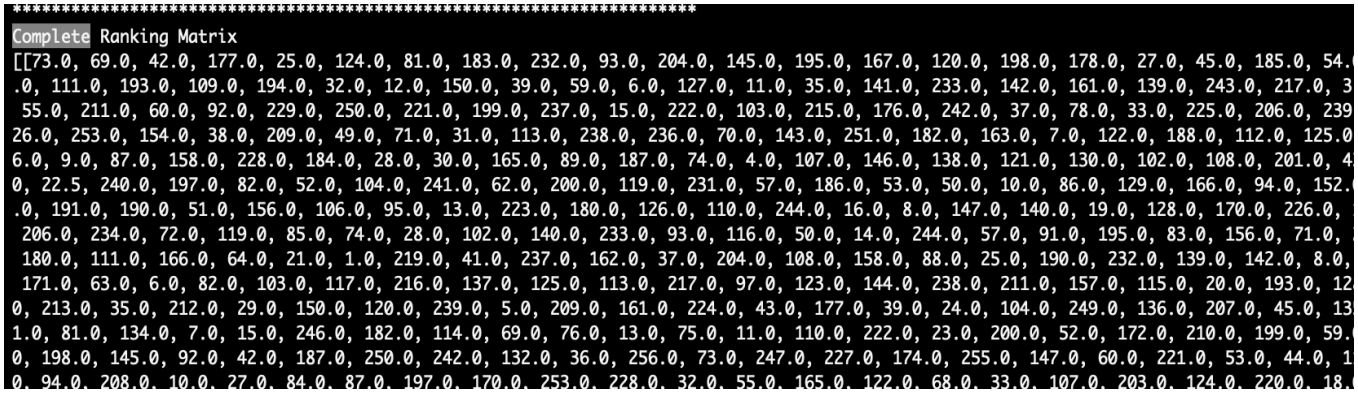


Figure 9: 4th Byte analysis Complete Rank Matrix

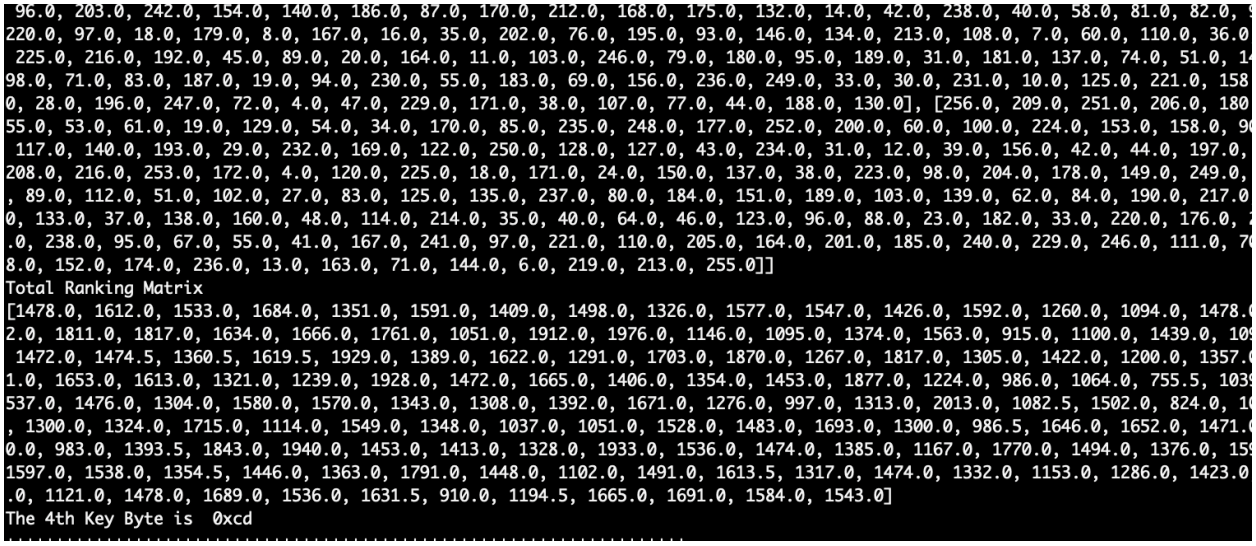


Figure 10: 4th Byte Output and Total Rank Matrix

OUTPUT For Finding 5th Byte Of Key:-

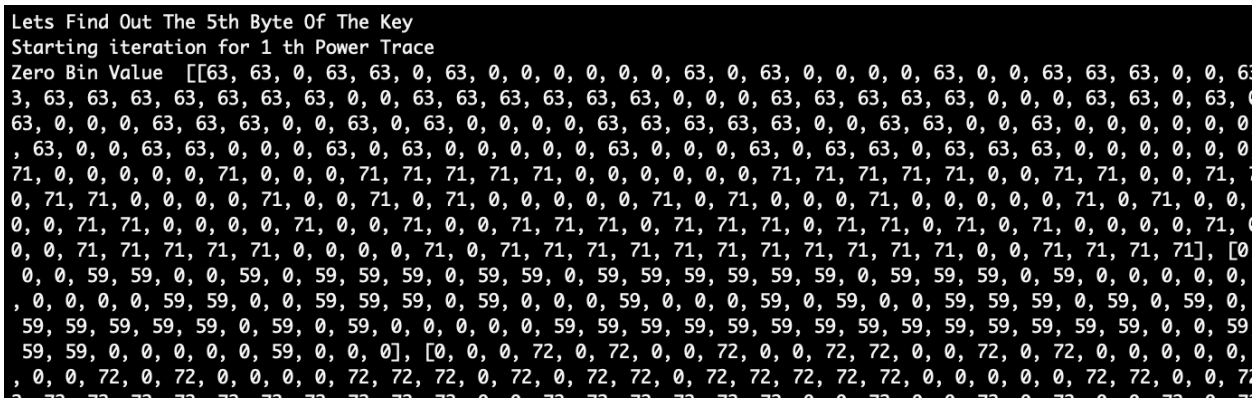


Figure 11: 5th Byte analysis with 1st Power Trace

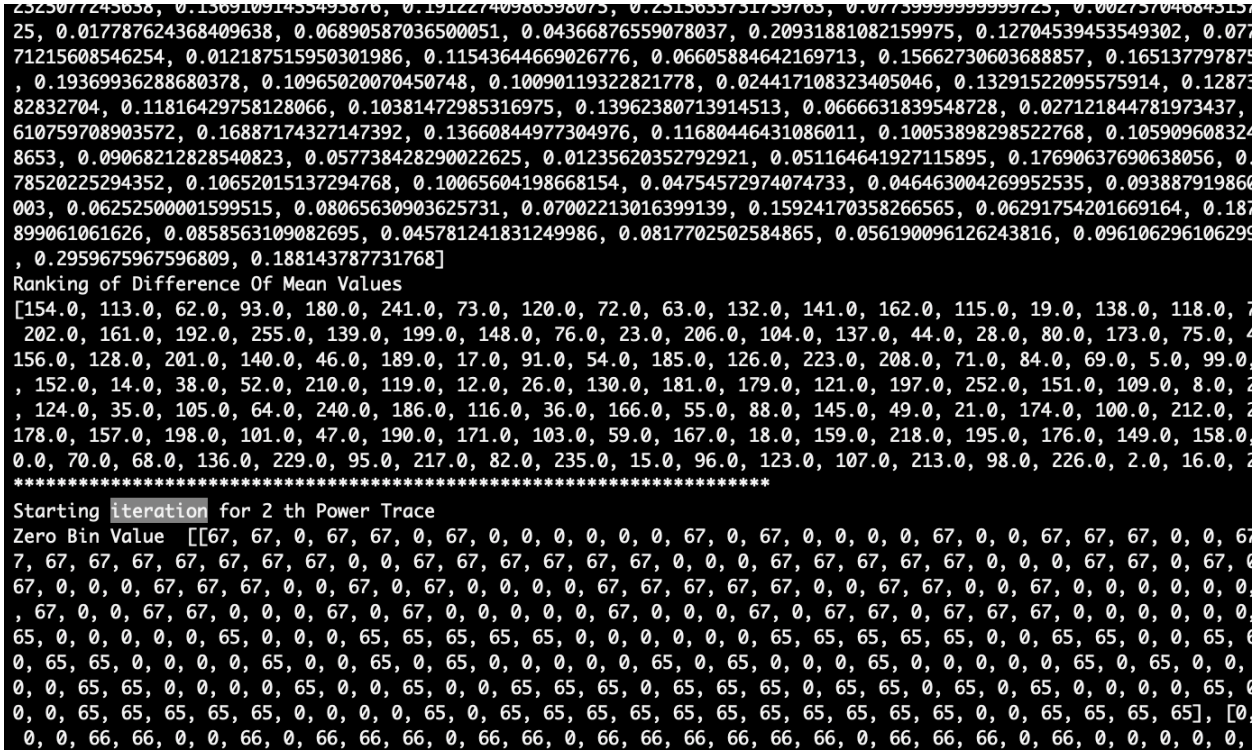


Figure 12: 5th Byte analysis with 2nd Power Trace

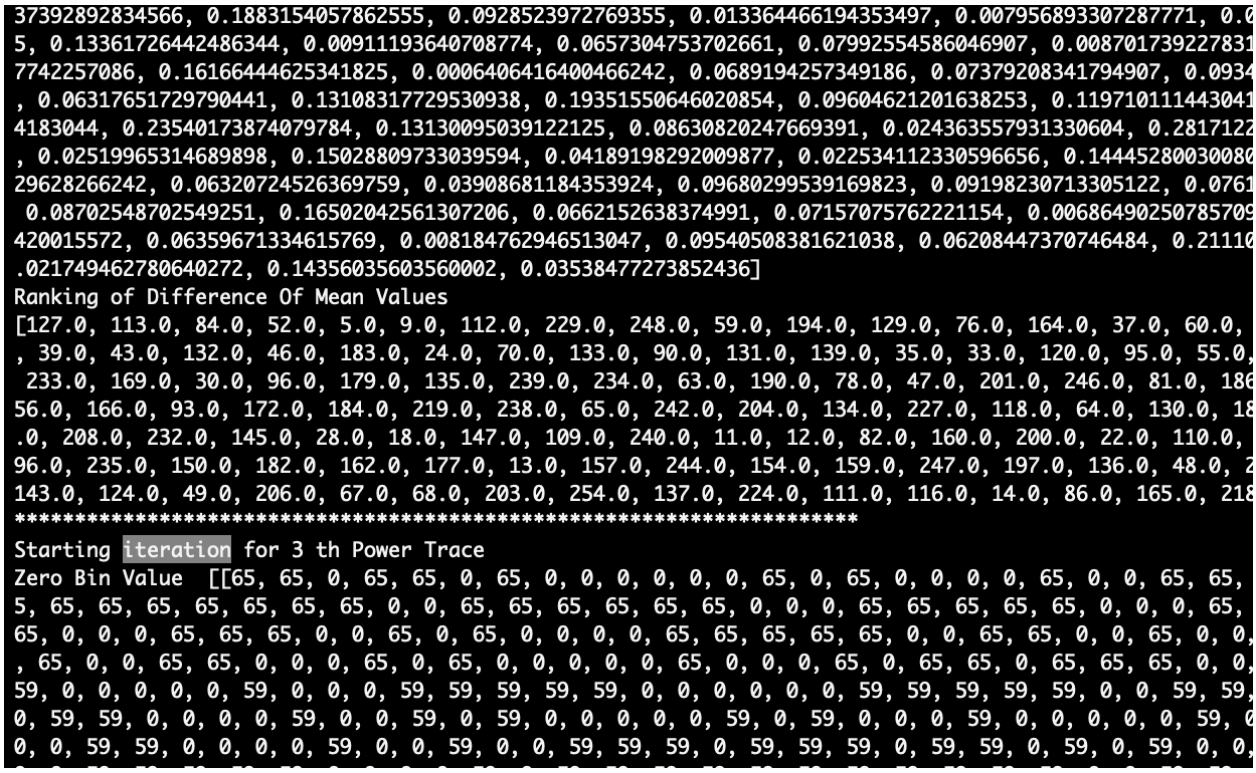


Figure 13: 5th Byte analysis with 3rd Power Trace

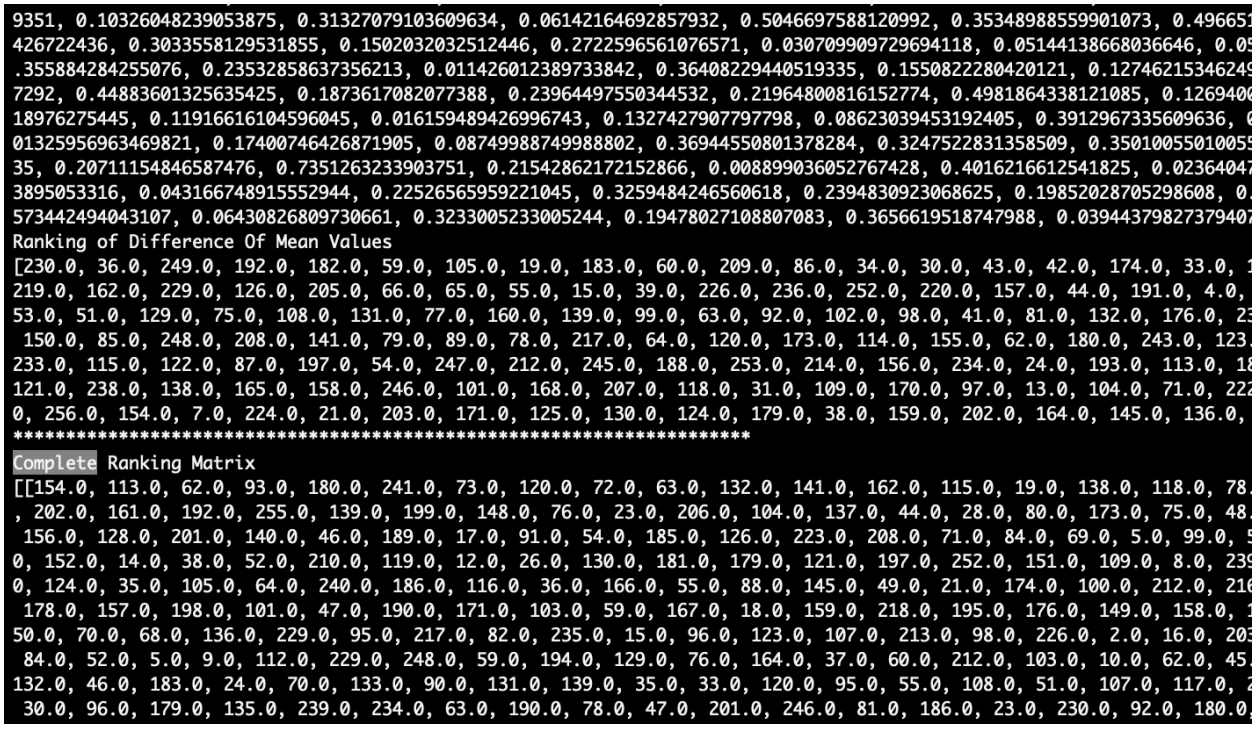


Figure 14: 5th Byte Complete Ranking Matrix

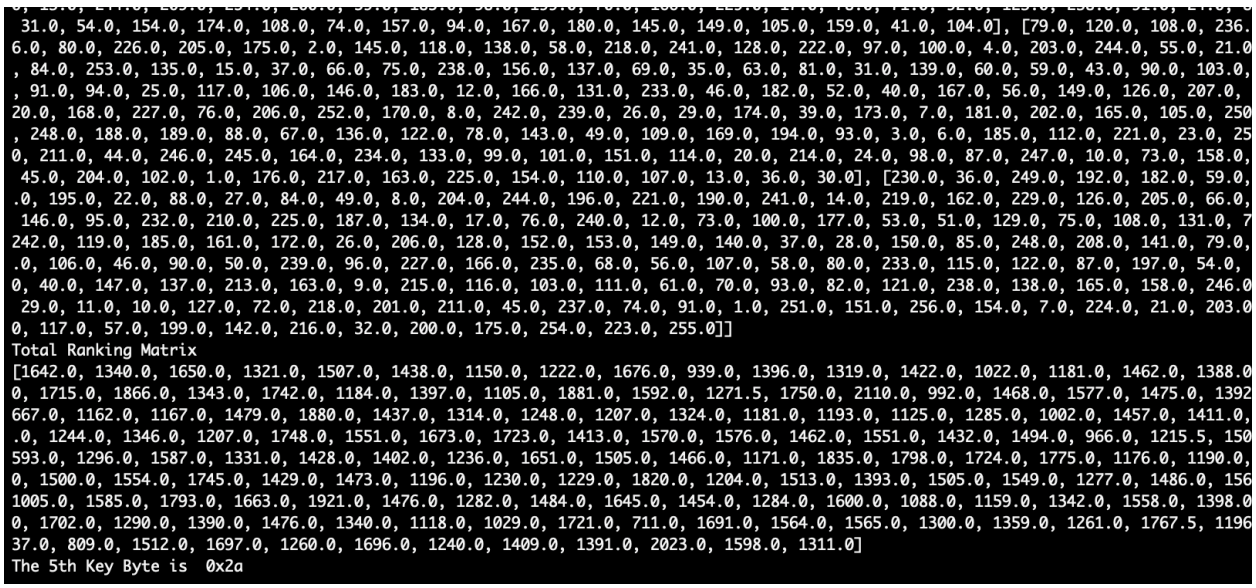


Figure 15: 5th Byte analysis Output And Total Ranking Matrix

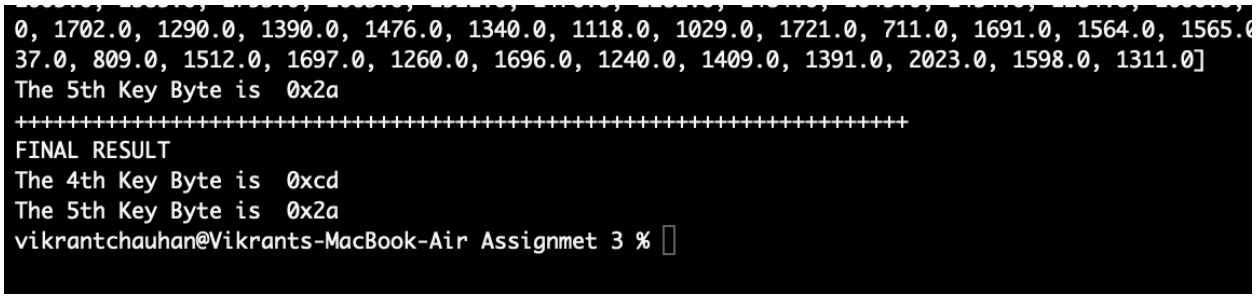


Figure 16: Final Output

Assignment 3, Question No. 2, Differential Fault Attack on AES

Group 2:

– Correct Ciphertext1: 0xb21eeb73953e7a2771db222ecbeea788

– Correct Ciphertext2: 0xcabe3e9988d6666a96a39e7b659ca91

– Faulty Ciphertext1: 0x33cf60141fd2f121ff9a6126fefd03e6

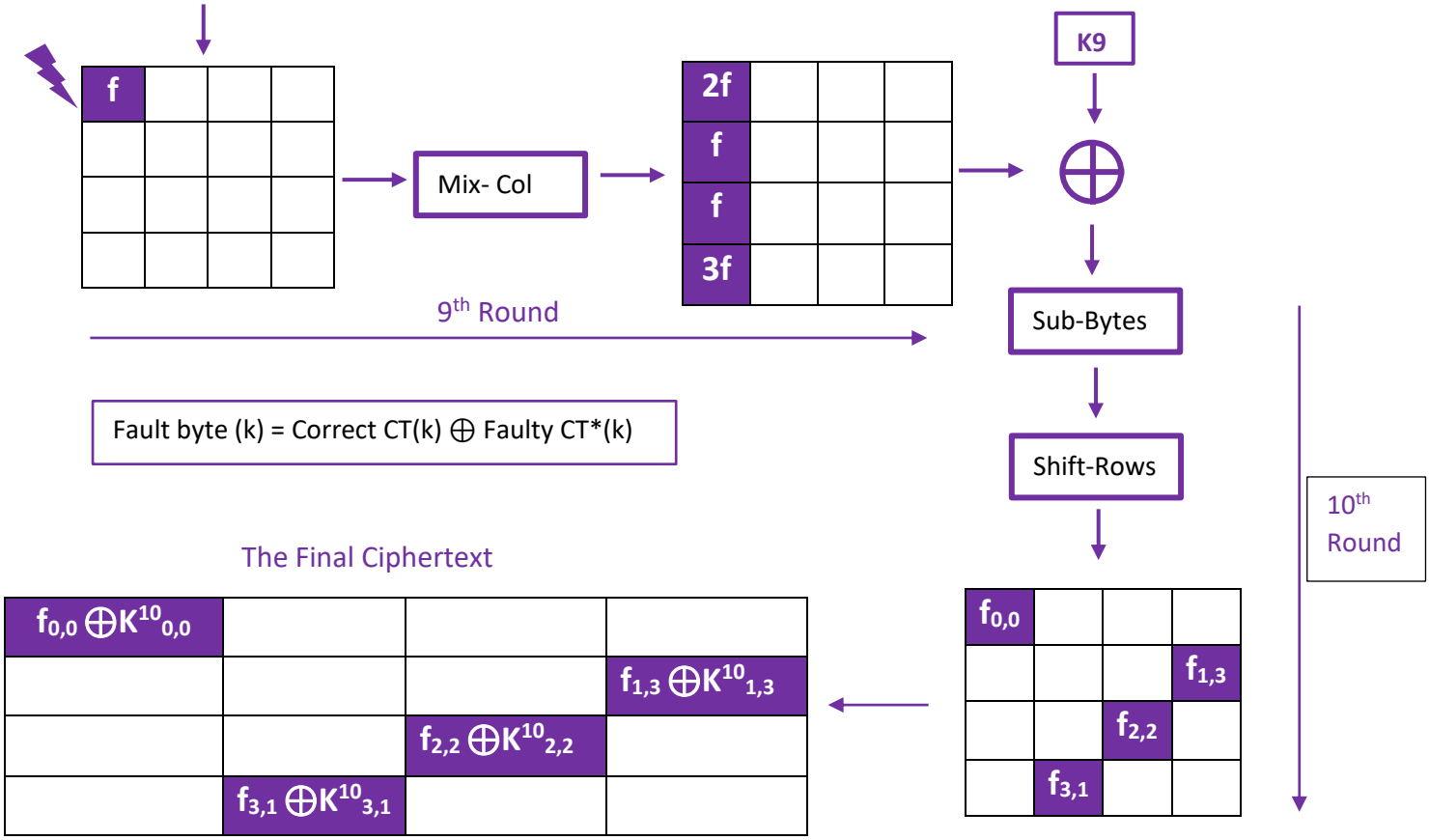
– Faulty Ciphertext2: 0x92317b8a9e24f1960f37385a4085b0d5

Problem and approach

We are provided with a pair of correct and faulty ciphertext, we are expected to formulate equations to recover one column of the key of round 10. This is the standard fault injection, injected in the start of the 9^{th} round in an AES, injected in the 0^{th} row and 0^{th} column Byte.

This fault is then expected to spread across an entire column after one round of the AES operation, with reverse engineering and publicly available algorithms of S-Box, we can formulate *inverse S-Box*, and we'll do the maths to reverse the *add round key* (using all possible guessed key for a Byte (i.e. 2^8 combinations)) we can do *shift rows* and *Mix-column* (using the irreducible polynomial theorem of GF(2^8) operation) to formulate 4 sets of equation.

The flow of equation generation/ Fault injection is as follows:



The formulated equations are:

Eqn1. $2f = S^{-1}(C_{0,0} \oplus K^{10}_{0,0}) \oplus S^{-1}(C^{*}_{0,0} \oplus K^{10}_{0,0})$

Eqn2. $f = S^{-1}(C_{1,3} \oplus K^{10}_{1,3}) \oplus S^{-1}(C^{*}_{1,3} \oplus K^{10}_{1,3})$

Eqn3. $f = S^{-1}(C_{2,2} \oplus K^{10}_{2,2}) \oplus S^{-1}(C^{*}_{2,2} \oplus K^{10}_{2,2})$

Eqn4. $3f = S^{-1}(C_{3,1} \oplus K^{10}_{3,1}) \oplus S^{-1}(C^{*}_{3,1} \oplus K^{10}_{3,1})$

So, we have created a python code who takes the two pairs of (C,C*) as input.

We then enumerate all possible keys for a Byte (i.e. for 8 bits, we get 2^8 possible keys, i.e. 256)

We then compute XOR of the (C and potential keys) and XOR of the (C* and potential keys)

We then compute inverse S-Box of the above two entities and compute the XOR of them.

We then form 6 set of equations from these 4 equations

(Eqn1=2*Eqn2),(Eqn1=2*Eqn3),(3*Eqn1=2*Eqn4),(2*Eqn2=Eqn3), (2*Eqn2=Eqn4), (3*Eqn3=Eqn4)

Here (*) is not multiplication, but is multiplication with irreducible polynomial of (GF(2^8): $X^8+X^4+X^3+X+1$)

Using these 6 set of equations, after round 1 (1st pair of C,C*) the potential keys, reduce from 256 to 30.

After the second round, we are left with our unique key column.

With given fault in the start of 9th round, the two pairs of CT and CT* and with the help of the 4 equations, we've successfully retrieved 4*(1B) of 10th round key (i.e. one column of key)

The extracted location of the key is ($K^{10}_{0,0}$, $K^{10}_{1,3}$, $K^{10}_{2,2}$, $K^{10}_{3,1}$)

We have followed a column major order throughout the code.

OUTPUT of our program:

```
C:\Users\souvik\OneDrive\Desktop> python q2.py

*******************************************************************************
 *****  Evaluation of step wise key-space reduction of target Bytes  *****
 *****  No. of potential keys in respective positions  *****
*******************************************************************************

Initially 2^8 keys possible for every Byte: k00: 256 |*|  k13: 256 |*| k22: 256 |*| k31: 256

*******************************************************************************

After Eqn 1: k00: 126  |*|  k13: 128  |*| k22: 256 |*| k31: 256

*******************************************************************************

After Eqn 2: k00: 62  |*|  k13: 128  |*| k22: 62  |*| k31: 256

*******************************************************************************
```

```
After Eqn 3: k00: 30  |*|  k13: 128  |*| k22: 62  |*| k31: 30

********************************************************************************

After Eqn 4: k00: 30  |*|  k13: 64  |*| k22: 62  |*| k31: 30

********************************************************************************

After Eqn 5: k00: 30  |*|  k13: 30  |*| k22: 62  |*| k31: 30

********************************************************************************

After Eqn 6: k00: 30  |*|  k13: 30  |*| k22: 30  |*| k31: 30

********************************************************************************

 *****  Second Iteration, with Correct & Faulty Ciphertext number 2 *****

********************************************************************************

After Eqn 7: k00: 3  |*|  k13: 3  |*| k22: 30  |*| k31: 30

********************************************************************************

After Eqn 8: k00: 1  |*|  k13: 3  |*| k22: 1  |*| k31: 30

********************************************************************************

After Eqn 9: k00: 1  |*|  k13: 3  |*| k22: 1  |*| k31: 1

********************************************************************************

After Eqn 10: k00: 1  |*|  k13: 1  |*| k22: 1  |*| k31: 1   Key Successfully Recovered

********************************************************************************

After Eqn 11: k00: 1  |*|  k13: 1  |*| k22: 1  |*| k31: 1

********************************************************************************

After Eqn 12: k00: 1  |*|  k13: 1  |*| k22: 1  |*| k31: 1

********************************************************************************

Final value of Bytes column 1 of key10 (in Hexadecimal) r0c0: 6d r1c3 83 r2c2 a3 r3c1 59

********************************************************************************
Final value of Bytes column 1 of key10 (in Binary) r0c0: [(0, 1, 1, 0, 1, 1, 0, 1)] r1c3 [(1,
0, 0, 0, 0, 0, 1, 1)] r2c2 [(1, 0, 1, 0, 0, 0, 1, 1)] r3c1 [(0, 1, 0,
1, 1, 0, 0, 1)]

********************************************************************************

PS C:\Users\souvik\OneDrive\Desktop>
```