**PHASE 1**

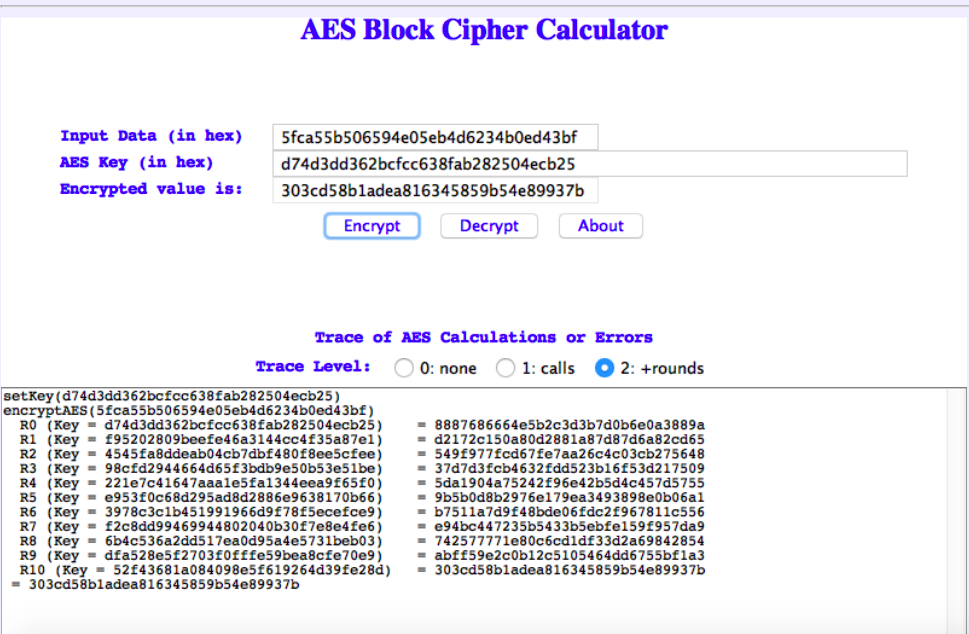
**TASK 1**

Given,

**Plain Text:** 5fca55b506594e05eb4d6234b0ed43bf

**Key:** d74d3dd362bcfcc638fab282504ecb25

Using the online AES calculator, we get the output:

****

The value of state after round 4 is: 5da1904a75242f96e42b5d4c457d5755.

**a)** The intial plain text is given as:

PlainText in Hex:

5fca55b506594e05eb4d6234b0ed43bf

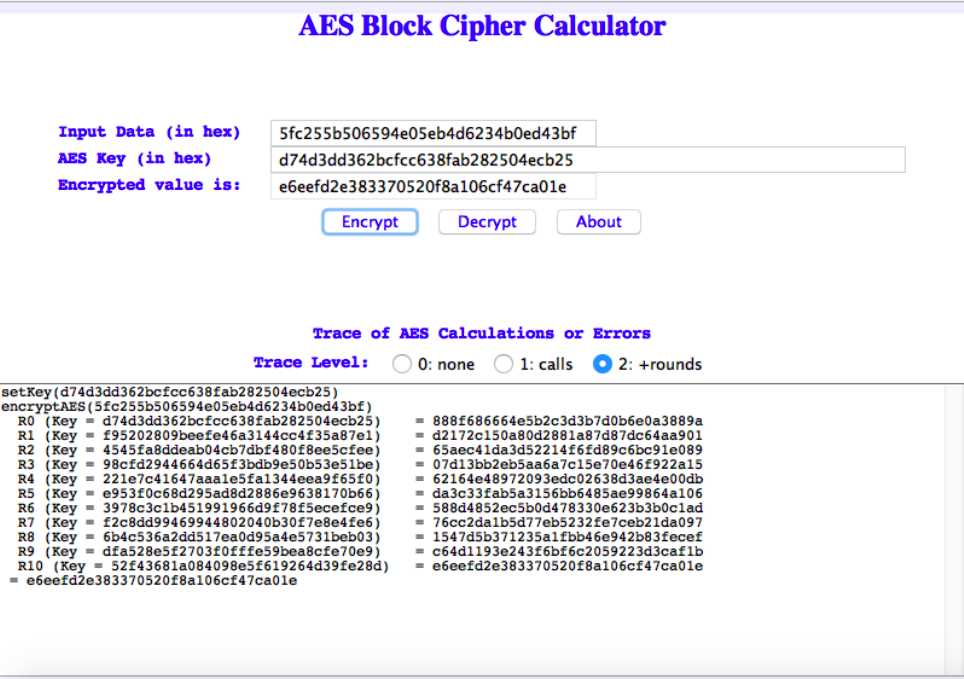
Binary after converting the 12th bit:

1011111110000100101010110110101000001100101100101001110000001011110101101001101011000100011010010110000111011010100001110111111

New Plaintext in Hex:

5FC255B506594E05EB4D6234B0ED43BF

AES Encryption after changing bit 12 of the plain text is calculated as:



Looking at the number of changed before and after Encryption.

|  |  |  |  |
| --- | --- | --- | --- |
| Rounds | With Original Plain Text | With New Plain Text | No. of bits changed |
| R0 | 8887686664E5B2C3D3B7D0B6E0A3889A | 888F686664E5B2C3D3B7D0B6E0A3889A | 1 |
| R1 | D217C150A80D2881A87D87D6A82CD65 | D2172C150A80D2881A87D87DC64AA901 | 28 |
| R2 | 549F977FCD67FE7AA26C4C03CB275648 | 65AEC41DA3D52214F6FD89C6BC91E089 | 42 |
| R3 | 37D7D3FCB4632FDD523B16F53D217509 | 07D13BB2EB5AA6A7C15E70E46F922A15 | 50 |
| R4 | 5DA1904A7524F96E42B5D4C457D5755 | 62164E48972093EDC0268D3AE4E00DB | 58 |

Fig shows the number of bits changed in each round with just one bit changed in Plaintext.

**b)** We can see that, even 1-bit change in the plaintext resulted in lot of bit changes in the encrypted text at each round. This is called the avalanche effect. This makes it very difficult for the cryptanalyst to figure out of the plaintext.

**Task 2**

The key provided is: d74d3dd362bcfcc638fab282504ecb25

The plaintext provided is: 5fca55b506594e05eb4d6234b0ed43bf

To determine the first 8 sub-key words, we split the key provided (128 bits) as follows:

k0= d7 4d 3d d3

k1= 62 bc fc c6

k2= 38 fa b2 82

k3= 50 4e cb 25

k4=k0⊕g(k3), k5=k1⊕k4, k6=k2⊕k5, k7=k3⊕k6

To determine g(k3) we perform the following steps:

1. RotWord performs a 1-byte circular shift, so k3 becomes (4e cb 25 50)
2. SubWord performs a byte substitution on each byte of the input using the 16x16(Hex) S-box given below:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 0 | 63 | 7c | 77 | 7b | f2 | 6b | 6f | c5 | 30 | 01 | 67 | 2b | fe | d7 | ab | 76 |
| 1 | ca | 82 | c9 | 7d | fa | 59 | 47 | f0 | ad | d4 | a2 | af | 9c | a4 | 72 | c0 |
| 2 | b7 | fd | 93 | 26 | 36 | 3f | f7 | Cc | 34 | a5 | e5 | f1 | 71 | d8 | 31 | 15 |
| 3 | 04 | c7 | 23 | c3 | 18 | 96 | 05 | 9a | 07 | 12 | 80 | e2 | eb | 27 | b2 | 75 |
| 4 | 09 | 83 | 2c | 1a | 1b | 6e | 5a | a0 | 52 | 3b | d6 | b3 | 29 | e3 | 2f | 84 |
| 5 | 53 | d1 | 00 | ed | 20 | Fc | b1 | 5b | 6a | cb | be | 39 | 4a | 4c | 58 | cf |
| 6 | d0 | ef | Aa | fb | 43 | 4d | 33 | 85 | 45 | f9 | 02 | 7f | 50 | 3c | 9f | a8 |
| 7 | 51 | a3 | 40 | 8f | 92 | 9d | 38 | f5 | bc | b6 | da | 21 | 10 | ff | f3 | d2 |
| 8 | cd | 0c | 13 | ec | 5f | 97 | 44 | 17 | c4 | a7 | 7e | 3d | 64 | 5d | 19 | 73 |
| 9 | 60 | 81 | 4f | dc | 22 | 2a | 90 | 88 | 46 | ee | b8 | 14 | de | 5e | 0b | db |
| A | e0 | 32 | 3a | 0a | 49 | 06 | 24 | 5c | c2 | d3 | ac | 62 | 91 | 95 | e4 | 79 |
| B | e7 | c8 | 37 | 6d | 8d | d5 | 4e | a9 | 6c | 56 | f4 | ea | 65 | 7a | ae | 08 |
| C | Ba | 78 | 25 | 2e | 1c | a6 | b4 | c6 | e8 | dd | 74 | 1f | 4b | bd | 8b | 8a |
| D | 70 | 3e | b5 | 66 | 48 | 03 | f6 | 0e | 61 | 35 | 57 | b9 | 86 | c1 | 1d | 9e |
| E | e1 | f8 | 98 | 11 | 69 | d9 | 8e | 94 | 9b | 1e | 87 | e9 | ce | 55 | 28 | df |
| F | 8c | a1 | 89 | 0d | bf | e6 | 42 | 68 | 41 | 99 | 2d | 0f | b0 | 54 | bb | 16 |

SubWord(k3) = (2f 1f 3f 53)

The results of step 1 and 2 are XORed with a round constant, Rcon[j] = (RC[j], 0, 0, 0) = (RC[1], 0, 0, 0) with RC[1] = 1 = 0X01

g(k3) = SubWord(k3) ⊕ Rcon[1] (j = 1 as it is the first round)

g(k3) = (2f 1f 3f 53) ⊕ (01 00 00 00)

As mentioned above, the rest of the values can be calculated as follows:

k4=k0 ⊕ g(k3) = (d7 4d 3d d3) ⊕ (2e 1f 3f 53) = (f9 52 02 80)

k5=k1⊕ k4 = (62 bc fc c6) ⊕ (f9 52 02 80) = (9b ee fe 46)

k6=k2 ⊕ k5 = (38 fa b2 82) ⊕ (9b ee fe 46) = (a3 14 4c c4)

k­7=k3 ⊕k6 = (50 4e cb 25) ⊕ (a3 14 4c c4) = (f3 5a 87 e1)

On performing the initial AddRound Key stage we need to XOR the key and plaintext provided.

**Key Plaintext New** **state**

d7 4d 3d d3 5f ca 55 b5 88 87 68 66

62 bc fc c6 06 59 4e 05 64 e5 b2 c3

38 fa b2 82 eb 4d 62 34 d3 b7 d0 b6

50 4e cb 25 b0 ed 43 bf e0 a3 88 9a

We can verify this value using the AES calculator. The new state can be verified after step R0.

**SubBytes stage, we determine the next state by looking up the S-box:**

c4 17 45 33

43 d9 37 2e

66 a9 70 4e

e1 0a c4 b8

**Shift row transformation**

c4 17 45 33

d9 37 2e 43

70 4e 66 a9

b8 e1 0a c4

**Mix Column Transformation**

02 03 01 01 c4 17 45 33

01 02 03 01 \* d9 37 2e 43

01 01 02 03 70 4e 66 a9

03 01 01 02 b8 e1 0a c4

S0,0 = {02}. {c4} ⊕ {03}. {d9} ⊕ {01}. {70} ⊕ {01}. {b8} = {02}. {c4}⊕ ({02}. {d9} ⊕ d9) ⊕ 70 ⊕ b8 = b2

S0,1 = {02}. {17} ⊕ {03}. {37} ⊕ {01}. {4e} ⊕ {01}. {e1} = {02}. {17} ⊕ ({02}. {37} ⊕ 37) ⊕ 4e ⊕ e1 = d8

S0,2 = {02}. {45} ⊕ {03}. {2e}⊕ {01}. {66} ⊕ {01}. {0a} = {02}. {45} ⊕ ({02}. {2e} ⊕ 2e) ⊕ 66 ⊕ 0a = 94

S0,3 = {02}. {33} ⊕ {03}. {43} ⊕ {01}. {a9} ⊕ {01}. {c4} = {02}. {33} ⊕ ({03}. {43} ⊕ 0a) ⊕ a9 ⊕ c4 = ce

S1,0 = {01}. {c4} ⊕ {02}. {d9} ⊕ {03}. {70} ⊕ {01}. {b8} = c4 ⊕ {02}. {d9} ⊕ ({02}. {70} ⊕ 70) ⊕ b8 = dc

S1,1 = {01}. {17} ⊕ {02}. {37} ⊕ {03}. {4e} ⊕ {01}. {e1} = 17 ⊕ {02}. {37} ⊕ ({02}. {4e} ⊕ b1) ⊕ e1 = 4a

S1,2 = {01}. {45} ⊕ {02}. {2e} ⊕ {03}. {66} ⊕ {01}. {0a} = 45 ⊕ {02}. {2e} ⊕ ({02}. {66} ⊕ 66) ⊕ 0a = b9

S1,3 = {01}. {33} ⊕ {02}. {43} ⊕ {03}. {a9} ⊕ {01}. {c4} = 33 ⊕ {02}. {43} ⊕ ({02}. {a9} ⊕ a9) ⊕ c4 = 91

S2,0 = {01}. {c4} ⊕ {01}. {d9} ⊕ {02}. {70} ⊕ {03}. {b8} = c4 ⊕ d9 ⊕ {02}. {70} ⊕ ({02}. {b8} ⊕ b8) = 2e

S2,1 = {01}. {17} ⊕ {01}. {37} ⊕ {02}. {4e} ⊕ {03}. {e1} = 17 ⊕ 37 ⊕ {02}. {4e} ⊕ ({02}. {e1} ⊕ e1) = 84

S2,2 = {01}. {45} ⊕ {01}. {2e} ⊕ {02}. {66} ⊕ {03}. {0a} = 45 ⊕ 2e ⊕ {02}. {66} ⊕ ({02}. {0a} ⊕ 0a) = b9

S2,3 = {01}. {33} ⊕ {01}. {43} ⊕ {02}. {a9} ⊕ {03}. {c4} = 33 ⊕ 43 ⊕ {02}. {a9} ⊕ ({02}. {c4} ⊕ c4) = 6e

S3,0 = {03}. {c4} ⊕ {01}. {d9} ⊕ {01}. {70} ⊕ {02}. {b8} = ({02}. {c4} ⊕ c4) ⊕ d9 ⊕ 70 ⊕ {02}. {b8} = 95

S3,1 = {03}. {17} ⊕ {01}. {37} ⊕ {01}. {4e} ⊕ {02}. {e1} = ({02}. {17} ⊕ 17) ⊕ 37 ⊕ 4e ⊕ {02}. {e1} = 99

S3,2 = {03}. {45} ⊕ {01}. {2e} ⊕ {01}. {66} ⊕ {02}. {0a} = ({02}. {45} ⊕ 45) ⊕ 2e ⊕ 66 ⊕ {02}. {0a} = 93

S3,3 = {03}. {33} ⊕ {01}. {43} ⊕ {01}. {a9} ⊕ {02}. {c4} = ({02}. {33} ⊕ 33) ⊕ 43⊕ a9 ⊕ {02}. {c4} = 2c

**State after mix column:**

b2 d8 94 ce

cd 4a b9 91

2e 84 b9 6e

95 99 93 2c

**State Sub-key for round 1 State after round 1**

b2 d8 94 ce f9 52 02 80 4b 8a 96 4e

cd 4a b9 91 9b ee fe 46 56 a4 47 d7

2e 84 b9 6e a3 14 4c c4 8d 90 f5 aa

95 99 93 2c f3 5a 87 e1 66 c3 14 cd

We can verify this value using the AES calculator. The new state can be verified after step R1.

**TASK 3**

**a) Cipher Block Chaining**

Key: VaishnaviKannanJ

Value of Key in Hexadecimal: 5661 6973 686e 6176 694b 616e 6e61 6e4a

Plaintext: This is a project for Computer Security Project2

Plaintext split into blocks of 16 characters each: Thisisaprojectfo rComputerSecurit yProject2

Plaintext in Hexadecimal:

PT1: 5468 6973 6973 6170 726f 6a65 6374 666f

PT2: 7243 6f6d 7075 7465 7253 6563 7572 6974

PT3: 7950 726f 6a65 6374 320a

In cipher block chaining (CBC), the input is taken to be the 16 byte block of plaintext for encryption. At each step, the cipher text of the previous encryption is XORed with the current plaintext block, making the cipher text block dependent on all plaintext blocks.

C i = AES K1 (P i XOR C i-1 )

C -1 = IV

Here, IV is taken to be all zeroes of length equal to the plaintext block. So,

IV: 0000000000000000000000000000000000

**1. Encryption:**

**First cycle of encryption:**

According to the given formula, the first cipher text is given by the formula:

C i = AES K1 (P i XOR C i-1 )

Now, C1 = AES K1 ( P1 XOR C0)

Now, P1 = PT1 XOR IV = PT1

We used the AES online calculator to encrypt the XOR of PT1 and IV, to get C1.

We get, C1 = cc84737f17a8c2e24d59fdd83cdb86c3

Second cycle of encryption:

PT2 XOR C1 = bec71c1267ddb6873f0a98bb49a9efb7

And C2= 234c 478d e912 bfe2 960f 98fa 6359 a331

**Third cycle of encryption:**

Since, PT3 is not of 16 bytes, we use PKCS5 padding to increase its length. The new elongated PT3 is given by:

PT3 = 7950726f6a656374320a080808080808

PT3 XOR C2= 5a1c 35e2 8377 dc96 a405 90f2 6b51 ab39

Therefore, C3 = c43d301283e79a379858e0e190d24d5d

**2. Decryption:**

**First Round of decryption:**

For decryption, we use the cipher text that is obtained in each cycle, decrypt it using the key and XOR the value with the previous cipher text.

So, C3 = c43d301283e79a379858e0e190d24d5d

Key = 5661 6973 686e 6176 694b 616e 6e61 6e4a

The decryption of C3 with Key Is: 5a1c35e28377dc96a40590f26b51ab39

The above output is XORed with C2 to get PT3.

PT3 = 7950726f6a656374320a080808080808.

We remove the padding bits to get PT3 = 7950726f6a656374320a

**Second Round of decryption**

C2= 234c 478d e912 bfe2 960f 98fa 6359 a331

The decrypted value is: bec71c1267ddb6873f0a98bb49a9efb7

and PT2 = 72436f6d707574657253656375726974

**Third Round of decryption**

C1 = cc84737f17a8c2e24d59fdd83cdb86c3

The decrypted value is: 5468697369736170726f6a656374666f

The value is XORed with IV to get PT1 = 5468697369736170726f6a656374666f

**b) Cipher Text Feedback**

In Cipher Text Feedback,the encryption is given by the formula:

Ci = PTi XOR AES(C i-1)

**1. Encryption:**

**First Round of encryption:**

C1 = PT1 XOR AES(C0)

The XOR is 356a1d455bad0df007986394ecbf69e1 and C1 = 6102743632de6c8075f709f18fcb0f8e.

**Second Round of encryption:**

The XOR in this round is 9abef2a38a71bc6a38e11a439d73cb38 and

C2= e8fd9dcefa04c80f4ab27f20e801a24c

**Third Round of encryption**

The final cipher is C3 = c95694497ffae65dcbe868506e529927

**2. Decryption:**

The decryption is given by : Pti = AES(C i-1) XOR (C I)

**First Round of Decryption:**

PT3 = AES(C2) XOR C3

Therefore, AES(C2) XOR C3 = b006e626159f8529f9e26058665a912f and PT3 = 7950726f6a656374320a080808080808. We remove the padding bits to get PT3 = 7950726f6a656374320a.

**Second round of decryption:**

AES(C1) XOR C2 = 9abef2a38a71bc6a38e11a439d73cb38 and

PT2 = 72436f6d707574657253656375726974

**Third round of decryption:**

AES(C0) XOR C1 = 356a1d455bad0df007986394ecbf69e1 and

PT1 = 5468697369736170726f6a656374666f

**PHASE 2**

**Section 1 :**

Let’s list down some properties of embedded systems:

Embedded systems have low power consumption, small size, rugged operating ranges, and low per-unit cost. This comes at the price of limited processing resources, which make them significantly more difficult to program and to interact with.

In RSA, we perform multiplication of two large prime numbers which makes RSA slower. Thus, to compensate with RSA’s slow performance compared to others. We divide the calculation into smaller repetitive methods, which make the decryption process faster. This method keeps in mind the limited processing resources of embedded systems and explains the reason of having decryption process as explained in section 3.

**Section 2 :**

From phase 1, we know that:

M=SH1+SH2

Where SH1 = M1AmodN, SH2=M2BmodN, A=q p-1 mod N and B=p q-1 mod N.

Let the faulty message be M2’.

Therefore, M would become

M = [(Cp t mod p )( q p-1 mod N) + (Cq u mod p)( p q-1 mod N)] where t=alpha and u=beta, Cp = Cmod p and Cq = C modq and C Is the ciphertext.

M= [(Cp t mod p )( q p-1 mod N) + M2( p q-1 mod N)] mod N

M’ = [(Cp t mod p )( q p-1 mod N) + M2’ ( p q-1 mod N)] mod N

M – M’ = (M2 – M2’) ( p q-1 mod N) mod N

= (M2 – M2’) ( p q-2 mod N) (p mod N) mod N

so, p can be obtained by calculating the gcd of M-M’ and N. The knowledge of ‘p’ can be used to obtain the private key set (p, q, d, phi(n)). This is a vulnerability where the attacker can easily guess the private key by calculating the difference in M values.

**Section 3:**

The reason why the encryption process in sec. 3 phase 2 is proposed for encryption devices is because the computational time for CRT exponentiation is less compared to direct exponentiation.

Given in the algorithm,

C1 = Mt mod p

C2 = Mu mod q

C = ( SXC1 + TXC2 ) mod n

where S = 1modp, S=0modq i.e. S=x.q

and T=1modq, T=0modp i.e. T=y.p

let C2’ be the faulty C2 value. Then, C’ = ( SXC1 + TXC2’ ) mod n.

Now, C – C’ = ( C2 – C2’ ).y.p.mod N

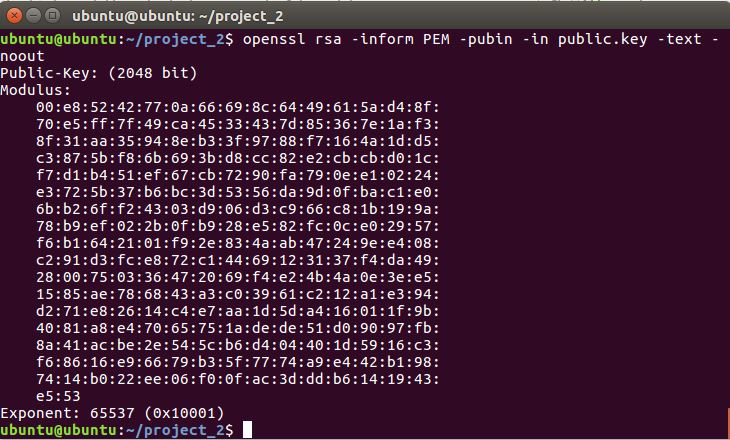
Now we can find the value of ‘p’ by taking the gcd(( C2 – C2’ ).y.p.mod N, pq). As in Section 2, the value of p will reveal the value of the secret key. This algorithm has the same vulnerability as the algorithm given in Section 2.

**Section 4:**

Given: C1 and C2.

The two major steps for decrypting the given file is:

1. Decrypt the given AES key, which has been encrypted using RSA.
2. Then, decrypt the file using the decrypted AES key.



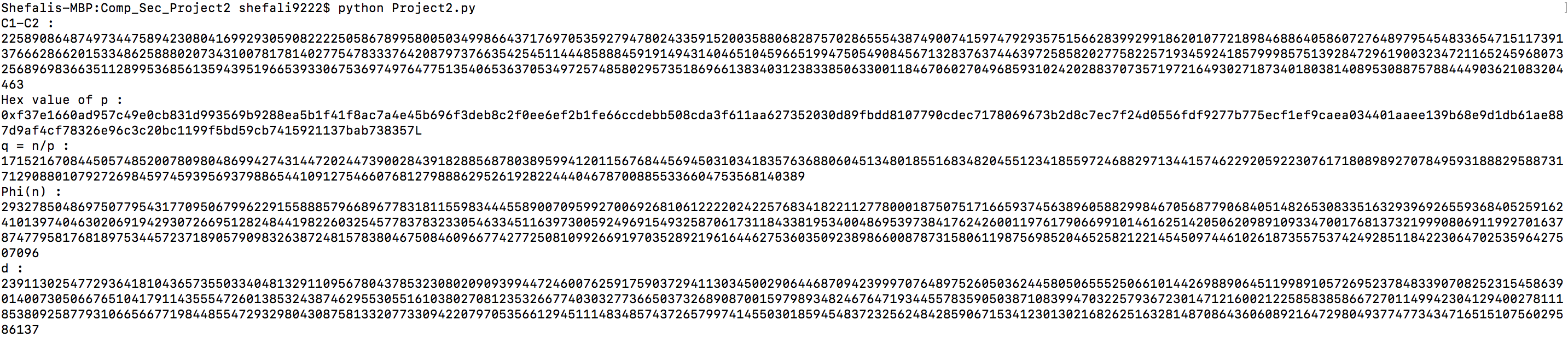
1. **Decryption of the given AES key.**

e= 65537.

n is extracted by converting the hexadecimal to decimal.

n=29327850486975077954317709506799622915588857966896778318115598344455890070959927006926810612222024225768341822112778000187507517166593745638960588299846705687790684051482653083351632939692655936840525916241013974046302069194293072669512824844198226032545778378323305463345116397300592496915493258706173118434162041944344570079998400688655706201477504735565673872643797590308549327458490185382239659524831890657732834868058859381713915190453642358738553777011410499179974986281138263624724940119570258010301077726098830442179627135846653352495979410539119788164279306621154419569604188731389339461710075619781810316627

1. **C1 and C2 and then converted into decimals and their difference is found out.**  
   **C1 =** ""  
     
   **C2 =** ""  
     
   **Difference:**225890864874973447589423080416992930590822225058678995800503499866437176970535927947802433591520035880682875702865554387490074159747929357515662839929918620107721898468864058607276489795454833654715117391376662866201533486258880207343100781781402775478333764208797376635425451144485888459191494314046510459665199475054908456713283763744639725858202775822571934592418579998575139284729619003234721165245968073256896983663511289953685613594395196653933067536974976477513540653637053497257485802957351869661383403123833850633001184670602704968593102420288370735719721649302718734018038140895308875788444903621083204463  
     
   **Given Cipher Text (AES Key-Encrpted):**C=3179740012625133057570999348796020887970038322732777228738492234531231820889307329189409692754561923056803476549939303529718689858702338843509879916826984613997453938464073455419919859561724235119815171967343259316816605736370751069002667018517825866898504185521662163831290151087168895657274298164106454476033402855485248343666287900812381327499471739362306857120107690923436034512063607579122064675524255486740345171955127712070661358079770890507106789610244928458921800548246009448158666622655288829368575471450216764713710647937499935145033242905602801423582283102052143812739996246746558184196761974680829657914
2. After finding the difference, we take out the P = GCD(C1-C2, n)  
   P = “170986268630524593060698330595030594979066518502032423698404813527398429696232747706314398875251385034491188630788536018589859817829031690495333150343650306759255388261182420807368127708191742859826861458350829251234345650071101293050901003529752448403126527895163171307835992038057434030548268761431814669143”
3. Phi(n) = (p-1)\*(q-1)  
   Phi\_n: 29327850486975077954317709506799622915588857966896778318115598344455890070959927006926810612222024225768341822112778000187507517166593745638960588299846705687790684051482653083351632939692655936840525916241013974046302069194293072669512824844198226032545778378323305463345116397300592496915493258706173118433819534004869539738417624260011976179066991014616251420506209891093347001768137321999080691199270163787477958176818975344572371890579098326387248157838046750846096677427725081099266919703528921961644627536035092389866008787315806119875698520465258212214545097446102618735575374249285118422306470253596427507096
4. We then Calculate d by using the function “modinv”:  
   d = “23911302547729364181043657355033404813291109567804378532308020909399447246007625917590372941130345002906446870942399970764897526050362445805065552506610144269889064511998910572695237848339070825231545863901400730506676510417911435554726013853243874629553055161038027081235326677403032773665037326890870015979893482467647193445578359050387108399470322579367230147121600212258583858667270114994230412940027811185380925877931066566771984485547293298043087581332077330942207970535661294511148348574372657997414550301859454837232562484285906715341230130216826251632814870864360608921647298049377477343471651510756029586137”  
     
   **We now have the required private key : (d,n)**



1. The following command can be used to find the standard notation of the public key:  
   **python Project2.py -f PEM -o key.txt –n –d**

Since we have the private and the public key, we can decrypt the AES key using openssl.

It is done so using the following command.

openssl rsautl -decrypt -inkey privatekey.pem -in key.bin.enc -out key.bin

Decrypted AES key is:

Decrypted AES Key in  decimal = 333277202522695828352578908493424296965

Key in hexadecimal = fabadababecafedeadbeef0102030405

**The command to decrypt the given file is:**

openssl enc -d -aes-128-ecb -in encrypted.aes -K fabadababecafedeadbeef0102030405 -out final.txt –nopad

**Python Code:  
  
**

**Reference Links:**

1. <https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet>
2. <http://searchsecurity.techtarget.com/definition/privilege-escalation-attack>
3. <http://www.angelfire.com/biz7/atleast/mix_columns.pdf>
4. <http://searchsecurity.techtarget.com/definition/cryptographic-checksum>
5. Stackoverflow
6. Cryptography and Network Security – Principles and Practices – 7th Edition – William Stallings