```
# 1- DES
input = [1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0
  , 0, 0, 1, 0, 1, 0, 0, 1, 1, 0]
round_key = [0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0,
  0, 0, 1, 0, 1, 1, 0, 1, 1, 1,
             1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1]
permutation_table = [16, 7, 20, 21, 29, 12, 28, 17,
                     1, 15, 23, 26, 5, 18, 31, 10,
                     2, 8, 24, 14, 32, 27, 3, 9,
                     19, 13, 30, 6, 22, 11, 4, 25]
expansion_table = [32, 1, 2, 3, 4, 5,
                     4, 5, 6, 7, 8, 9,
                     8, 9, 10, 11, 12, 13,
                     12, 13, 14, 15, 16, 17,
                     16, 17, 18, 19, 20, 21,
                     20, 21, 22, 23, 24, 25,
                     24, 25, 26, 27, 28, 29,
                     28, 29, 30, 31, 32, 1]
S_boxes = \
    [   [14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7], #S1
        [0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8],
        [4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0],
        [15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13]],\
    [   [15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10], #S2
        [3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5],
        [0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15],
        [13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9]],\
    [   [10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8], #S3
        [13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1],
        [13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7],
        [1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12],],\
    [   [7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15],  #S4
        [13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9],
        [10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4],
        [3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14],],\
    [   [2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9],  #S5
        [14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6],
        [4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14],
        [11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3],],\
    [   [12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11],  #S6
        [10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8],
        [9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6],
        [4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13],],\
    [   [4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1],   #S7
        [13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6],
        [1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2],
        [6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12],],\
    [   [13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7],  #S8
        [1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2],
```

```
48          [7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8],
49          [2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11],]
50 def dec2bin(n):
51     a=[]
52     a[:0] = format(n,'04b')
53     b=[]
54     for x in a:
55         b.append(int(x))
56     return b
57
58 # 1a- Extend the input to 48 bits using DES expansion function.
59 input_extended = [input[index-1] for index in expansion_table]
60 print('Extended input is :\n', input_extended)
61 '''
62 Output: Extended input is :
63 [0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1,
   1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1]
64 '''
65
66 # 1b- Add (XOR) the given round key to the expanded input bits.
67 input_xored = [input_extended[i] ^ round_key[i] for i in range(len(
   input_extended))]
68 print('XORed input with round key:\n',input_xored)
69 '''
70 Output: XORed input with round key:
71  [0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1
   , 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0]
72 '''
73 # 1c- Using 8 DES S-boxes, find the 32-bit output of substitution step.
   DES S-boxes are
74 # presented in the DES paper, appendix 1 (pages 17-18).
75 blocks = [input_xored[i:i+6] for i in range(0,48,6)]
76 print('Inputs to the S boxes:\n',blocks)
77 s_out = []
78 for s, block in enumerate(blocks):
79     row = block[0]*2+block[5]
80     column = block[1]*8 + block[2]*4 + block[3]*2 + block[4]
81     s_out.extend(dec2bin(S_boxes[s][row][column]))
82 print('Sbox output is:\n',s_out)
83 '''
84 Output:
85
86 Sbox output is:
87  [0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1
   , 0, 1, 1, 0, 0, 0, 1, 0]
88 '''
89 # 1d- Permute the S-box output using the given permutation table.
90 output_permuted = [s_out[idx-1] for idx in permutation_table]
91 print('Permuted output is:\n',output_permuted)
```

```
92 '''
93 Output:
94 Permuted output is:
95  [0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0
   , 1, 0, 0, 0, 1, 0, 0, 0]
96
97 '''
98 # 2 - Compute the given steps below. Please refer to the AES
   specification for more details.
99 # Show your work and present the results in a table to make it easy to
   follow.
100 input = [0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1,
   0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0,
101          0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1,
   0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1,
102          1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1,
   1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0,
103          1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0,
   0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1]
104 S_box = [[0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01,
   0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76],
105          [0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4,
   0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0],
106          [0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5,
   0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15],
107          [0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12,
   0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75],
108          [0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b,
   0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84],
109          [0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb,
   0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf],
110          [0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9,
   0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8],
111          [0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6,
   0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2],
112          [0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7,
   0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73],
113          [0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee,
   0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb],
114          [0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3,
   0xac, 0x62, 0x91, 0x95, 0xe4, 0x79],
115          [0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56,
   0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08],
116          [0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd,
   0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a],
117          [0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35,
   0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e],
118          [0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e,
   0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf],
```

```
119         [0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99,
    0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16]]
120 round_key = [0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0,
     0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0,
121              0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0,
     1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1,
122              1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
     1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1,
123              1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1,
     0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0]
124 round_key_hex=[[0x34, 0x09, 0xa6, 0xd6],
125               [0x76, 0x93, 0x28, 0x43],
126               [0xd5, 0x04, 0xc8, 0xcd],
127               [0xf1, 0xb5, 0x72, 0x72]]
128 irreducable = [1,0,0,0,1,1,0,1,1]
129 def get_columns(matrix):
130     columns = [[matrix[j][i] for j in range(len(matrix))] for i in range
    (len(matrix[0]))]
131     return columns
132 def xor(list1,list2):
133     assert len(list1)==len(list2)
134     res=[]
135     for k in range(len(list1)):
136         res.append(list1[k]^list2[k])
137     return res
138 def reduce_poly(result,irreducable):
139     irr=irreducable[:]
140     [irr.append([0]) for k in range(len(result)-9)]
141     temp=result[:]
142     for i in range(len(result)-8):
143         temp = xor(temp,irr)
144         irr.pop(len(irr)-1)
145         irr.insert(0,0)
146     return temp
147 def gal_mul(column,coefficient):
148     out = [0]*8
149     for idx, element in enumerate(column):
150         c=[]
151         c[:0] = format(int(element, 16),'08b')
152         d = [int(k) for k in c ]
153         c = bin(coefficient[idx])[2:]
154         c = [int(k) for k in c]
155         result = [0] * (len(d) + len(c) - 1)
156         for o1, i1 in enumerate(d):
157             for o2, i2 in enumerate(c):
158                 result[o1 + o2] += i1 * i2
159         result=[x%2 for x in result]
160         t = []
161         flag = 0
```

```
        #remove zeros
        for k in range(len(result)):
            if len(result) < 9: break
            if flag == 0:
                if result[k] != 0:
                    t.append(result[k])
                    flag = 1
            else:
                t.append(result[k])

        if len(t)!=0: result=t[:]
        if len(result)>8:
            result = reduce_poly(result,irreducable)
        # pad zeros
        if len(result)<8:
            for k in range(8 - len(result)): result.insert(0,0)
        out=xor(result[len(result)-8:len(result)],out)
    return out
# 2a- Convert the given 128-bit input to Hexadecimal form.
inp=''.join(map(str, input))
input_h = hex(int(inp, 2))
print('Input in Hexadecimal form:\n',input_h)
'''
Output:
Input in Hexadecimal form:
0x56e219b244b3db43811e9d3a9e85f34f
'''
# 2b- Write the input in a state diagram (4 by 4 matrix).
input_hex = [[0x56, 0xe2, 0x19, 0xb2],
             [0x44, 0xb3, 0xdb, 0x43],
             [0x81, 0x1e, 0x9d, 0x3a],
             [0x9e, 0x85, 0xf3, 0x4f]]
# 2c- Apply SubBytes Step: use AES S-box to substitute the input.
SubBytes_out=[]
for x,r in enumerate(input_hex):
    out_row=[]
    for y,c in enumerate(r):
        column = int(c%16)
        row = int((c - column)/16)
        out_row.append(hex(S_box[row][column]))
    SubBytes_out.append(out_row)
print('\nAfter SubBytes Step:')
[print(row) for row in SubBytes_out]
'''
Output:
After SubBytes Step:

['0xb1', '0x98', '0xd4', '0x37']
['0x1b', '0x6d', '0xb9', '0x1a']
```

```
['0x0c', '0x72', '0x5e', '0x80']
['0x0b', '0x97', '0x0d', '0x84']
'''
# 2d- Apply ShiftRows Step.
ShiftRows_out = SubBytes_out[:]
for r,row in enumerate(ShiftRows_out):
    for k in range(r):
        temp = row.pop(0)
        row.append(temp)
    ShiftRows_out[r] = row
print('\nAfter ShiftRows Step:')
[print(row) for row in ShiftRows_out]
'''
Output:
After ShiftRows Step:

['0xb1', '0x98', '0xd4', '0x37']
['0x6d', '0xb9', '0x1a', '0x1b']
['0x5e', '0x80', '0xc', '0x72']
['0x84', '0x0b', '0x97', '0x0d']
'''

# 2e- Apply Mixcolumns Step: use Irreducible polynomial P(x)=x^8+x^4+x^3+x+1.
c_matrix = [[2, 3, 1, 1],
            [1, 2, 3, 1],
            [1, 1, 2, 3],
            [3, 1, 1, 2]]
columns = get_columns(ShiftRows_out)
Mixcolumns_out = ShiftRows_out[:]
temp=[]
for c, column in enumerate(columns):
    col=[]
    for j in range(4):
        col.append(gal_mul(column, c_matrix[j]))
    temp.append(col)
for k,r in enumerate(temp):
    for i,h in enumerate(r):
        h = ''.join(map(str, h))
        temp[k][i] = hex(int(h, 2))
Mixcolumns_out = get_columns(temp)
print('\nAfter MixColumns step:\n')
[print(row) for row in Mixcolumns_out]
'''
After MixColumns step:
Output:
['0x14', '0x70', '0x06', '0x3c']
['0x0d', '0x61', '0x63', '0x9a']
['0xf7', '0x27', '0x74', '0xdf']
```

```
['0xe8', '0x9c', '0x44', '0x2a']
'''

# 2f- Apply AddRoundKey Step: use the given round key
AddRoundKey_out=Mixcolumns_out[:]
for k,element in enumerate(Mixcolumns_out):
    row=[int(element[k], 16) for k in range(len(element))]
    AddRoundKey_out[k]= xor(round_key_hex[k],row)
for k,element in enumerate(AddRoundKey_out):
    row = [hex(element[k]) for k in range(len(element))]
    AddRoundKey_out[k]=row
print('\nAfter AddRoundKey step:\n')
[print(row) for row in AddRoundKey_out]
'''
Output:
After AddRoundKey step:

['0x20', '0x79', '0xa0', '0xea']
['0x7b', '0xf2', '0x4b', '0xd9']
['0x22', '0x23', '0xbc', '0x12']
['0x19', '0x29', '0x36', '0x58']
'''
# 3a AND 3b are below.
# 3c- List  all elements of modulo  216 with no multiplicative  inverse.
def gcd(n1, n2):
    if n2 == 0:
        return n1
    return gcd(n2, n1 % n2)

non_inv=[k for k in range(216) if gcd(k,216)!=1 ]
print(non_inv)

'''
Output:
[0, 2, 3, 4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, 21, 22, 24, 26, 27, 28
, 30, 32, 33, 34, 36, 38, 39, 40, 42, 44, 45, 46, 48, 50, 51, 52, 54, 56
, 57, 58, 60, 62, 63, 64, 66, 68, 69, 70, 72, 74, 75, 76, 78, 80, 81, 82
, 84, 86, 87, 88, 90, 92, 93, 94, 96, 98, 99, 100, 102, 104, 105, 106,
108, 110, 111, 112, 114, 116, 117, 118, 120, 122, 123, 124, 126, 128,
129, 130, 132, 134, 135, 136, 138, 140, 141, 142, 144, 146, 147, 148,
150, 152, 153, 154, 156, 158, 159, 160, 162, 164, 165, 166, 168, 170,
171, 172, 174, 176, 177, 178, 180, 182, 183, 184, 186, 188, 189, 190,
192, 194, 195, 196, 198, 200, 201, 202, 204, 206, 207, 208, 210, 212,
213, 214]
'''
```

3) a) i) $37.3 \bmod 23 = 14.3 \bmod 23$

$= 42 . \bmod 23 = \boxed{19 \bmod 23}$

ii) $19.13 \bmod 23 = (-4)(-10) \bmod 23$

$= \boxed{17 \bmod 23}$

iii) $18.15 \bmod 12 = 6.3 \bmod 12$

$= \boxed{6 \bmod 12}$

iv) $15.29 + 11.15 \bmod 23 = (-8).6 + (-12)(-8) \bmod 23$

$= 48 \bmod 23 = \boxed{2 \bmod 23}$

b) i) $8^{-1} \bmod 17 = a$

$a \cdot 8 \bmod 17 = 1 \bmod 17 = -16 \bmod 17 \Rightarrow a = -2 \bmod 17$

$= \boxed{15 \bmod 17}$

ii) $5^{-1} \bmod 17 = a$

$a\,5 \bmod 17 = 1 \bmod 17 = 35 \bmod 17 \Rightarrow \boxed{a = 7 \bmod 17}$

iii) $5^{-1} \bmod 37 = a$

$\gcd(37, 5) = 1$

$37 = 5 \cdot 7 + 2$

$5 = 2 \cdot 2 + 1$

$1 = 5 - 2 \cdot 2$

$1 = 5 - 2 \cdot (37 - 5 \cdot 7)$

$1 = 5 - 2(37) + 14 \cdot 5$

$1 = -2 \cdot 37 + 15 \cdot 5$

$1 \bmod 37 = 15 \cdot 5 \bmod 37$

$\boxed{a = 15}$

iv) $\gcd(10, 15) = 5 \neq 1$

There is no inverse of 10 in mod 15.