Name: Faraz Pathan

Rollno: COTB70

#AES

aes\_sbox = [

    [int('63', 16), int('7c', 16), int('77', 16), int('7b', 16), int('f2', 16), int('6b', 16), int('6f', 16), int('c5', 16), int(

        '30', 16), int('01', 16), int('67', 16), int('2b', 16), int('fe', 16), int('d7', 16), int('ab', 16), int('76', 16)],

    [int('ca', 16), int('82', 16), int('c9', 16), int('7d', 16), int('fa', 16), int('59', 16), int('47', 16), int('f0', 16), int(

        'ad', 16), int('d4', 16), int('a2', 16), int('af', 16), int('9c', 16), int('a4', 16), int('72', 16), int('c0', 16)],

    [int('b7', 16), int('fd', 16), int('93', 16), int('26', 16), int('36', 16), int('3f', 16), int('f7', 16), int('cc', 16), int(

        '34', 16), int('a5', 16), int('e5', 16), int('f1', 16), int('71', 16), int('d8', 16), int('31', 16), int('15', 16)],

    [int('04', 16), int('c7', 16), int('23', 16), int('c3', 16), int('18', 16), int('96', 16), int('05', 16), int('9a', 16), int(

        '07', 16), int('12', 16), int('80', 16), int('e2', 16), int('eb', 16), int('27', 16), int('b2', 16), int('75', 16)],

    [int('09', 16), int('83', 16), int('2c', 16), int('1a', 16), int('1b', 16), int('6e', 16), int('5a', 16), int('a0', 16), int(

        '52', 16), int('3b', 16), int('d6', 16), int('b3', 16), int('29', 16), int('e3', 16), int('2f', 16), int('84', 16)],

    [int('53', 16), int('d1', 16), int('00', 16), int('ed', 16), int('20', 16), int('fc', 16), int('b1', 16), int('5b', 16), int(

        '6a', 16), int('cb', 16), int('be', 16), int('39', 16), int('4a', 16), int('4c', 16), int('58', 16), int('cf', 16)],

    [int('d0', 16), int('ef', 16), int('aa', 16), int('fb', 16), int('43', 16), int('4d', 16), int('33', 16), int('85', 16), int(

        '45', 16), int('f9', 16), int('02', 16), int('7f', 16), int('50', 16), int('3c', 16), int('9f', 16), int('a8', 16)],

    [int('51', 16), int('a3', 16), int('40', 16), int('8f', 16), int('92', 16), int('9d', 16), int('38', 16), int('f5', 16), int(

        'bc', 16), int('b6', 16), int('da', 16), int('21', 16), int('10', 16), int('ff', 16), int('f3', 16), int('d2', 16)],

    [int('cd', 16), int('0c', 16), int('13', 16), int('ec', 16), int('5f', 16), int('97', 16), int('44', 16), int('17', 16), int(

        'c4', 16), int('a7', 16), int('7e', 16), int('3d', 16), int('64', 16), int('5d', 16), int('19', 16), int('73', 16)],

    [int('60', 16), int('81', 16), int('4f', 16), int('dc', 16), int('22', 16), int('2a', 16), int('90', 16), int('88', 16), int(

        '46', 16), int('ee', 16), int('b8', 16), int('14', 16), int('de', 16), int('5e', 16), int('0b', 16), int('db', 16)],

    [int('e0', 16), int('32', 16), int('3a', 16), int('0a', 16), int('49', 16), int('06', 16), int('24', 16), int('5c', 16), int(

        'c2', 16), int('d3', 16), int('ac', 16), int('62', 16), int('91', 16), int('95', 16), int('e4', 16), int('79', 16)],

    [int('e7', 16), int('c8', 16), int('37', 16), int('6d', 16), int('8d', 16), int('d5', 16), int('4e', 16), int('a9', 16), int(

        '6c', 16), int('56', 16), int('f4', 16), int('ea', 16), int('65', 16), int('7a', 16), int('ae', 16), int('08', 16)],

    [int('ba', 16), int('78', 16), int('25', 16), int('2e', 16), int('1c', 16), int('a6', 16), int('b4', 16), int('c6', 16), int(

        'e8', 16), int('dd', 16), int('74', 16), int('1f', 16), int('4b', 16), int('bd', 16), int('8b', 16), int('8a', 16)],

    [int('70', 16), int('3e', 16), int('b5', 16), int('66', 16), int('48', 16), int('03', 16), int('f6', 16), int('0e', 16), int(

        '61', 16), int('35', 16), int('57', 16), int('b9', 16), int('86', 16), int('c1', 16), int('1d', 16), int('9e', 16)],

    [int('e1', 16), int('f8', 16), int('98', 16), int('11', 16), int('69', 16), int('d9', 16), int('8e', 16), int('94', 16), int(

        '9b', 16), int('1e', 16), int('87', 16), int('e9', 16), int('ce', 16), int('55', 16), int('28', 16), int('df', 16)],

    [int('8c', 16), int('a1', 16), int('89', 16), int('0d', 16), int('bf', 16), int('e6', 16), int('42', 16), int('68', 16), int(

        '41', 16), int('99', 16), int('2d', 16), int('0f', 16), int('b0', 16), int('54', 16), int('bb', 16), int('16', 16)]

]

reverse\_aes\_sbox = [

    [int('52', 16), int('09', 16), int('6a', 16), int('d5', 16), int('30', 16), int('36', 16), int('a5', 16), int('38', 16), int(

        'bf', 16), int('40', 16), int('a3', 16), int('9e', 16), int('81', 16), int('f3', 16), int('d7', 16), int('fb', 16)],

    [int('7c', 16), int('e3', 16), int('39', 16), int('82', 16), int('9b', 16), int('2f', 16), int('ff', 16), int('87', 16), int(

        '34', 16), int('8e', 16), int('43', 16), int('44', 16), int('c4', 16), int('de', 16), int('e9', 16), int('cb', 16)],

    [int('54', 16), int('7b', 16), int('94', 16), int('32', 16), int('a6', 16), int('c2', 16), int('23', 16), int('3d', 16), int(

        'ee', 16), int('4c', 16), int('95', 16), int('0b', 16), int('42', 16), int('fa', 16), int('c3', 16), int('4e', 16)],

    [int('08', 16), int('2e', 16), int('a1', 16), int('66', 16), int('28', 16), int('d9', 16), int('24', 16), int('b2', 16), int(

        '76', 16), int('5b', 16), int('a2', 16), int('49', 16), int('6d', 16), int('8b', 16), int('d1', 16), int('25', 16)],

    [int('72', 16), int('f8', 16), int('f6', 16), int('64', 16), int('86', 16), int('68', 16), int('98', 16), int('16', 16), int(

        'd4', 16), int('a4', 16), int('5c', 16), int('cc', 16), int('5d', 16), int('65', 16), int('b6', 16), int('92', 16)],

    [int('6c', 16), int('70', 16), int('48', 16), int('50', 16), int('fd', 16), int('ed', 16), int('b9', 16), int('da', 16), int(

        '5e', 16), int('15', 16), int('46', 16), int('57', 16), int('a7', 16), int('8d', 16), int('9d', 16), int('84', 16)],

    [int('90', 16), int('d8', 16), int('ab', 16), int('00', 16), int('8c', 16), int('bc', 16), int('d3', 16), int('0a', 16), int(

        'f7', 16), int('e4', 16), int('58', 16), int('05', 16), int('b8', 16), int('b3', 16), int('45', 16), int('06', 16)],

    [int('d0', 16), int('2c', 16), int('1e', 16), int('8f', 16), int('ca', 16), int('3f', 16), int('0f', 16), int('02', 16), int(

        'c1', 16), int('af', 16), int('bd', 16), int('03', 16), int('01', 16), int('13', 16), int('8a', 16), int('6b', 16)],

    [int('3a', 16), int('91', 16), int('11', 16), int('41', 16), int('4f', 16), int('67', 16), int('dc', 16), int('ea', 16), int(

        '97', 16), int('f2', 16), int('cf', 16), int('ce', 16), int('f0', 16), int('b4', 16), int('e6', 16), int('73', 16)],

    [int('96', 16), int('ac', 16), int('74', 16), int('22', 16), int('e7', 16), int('ad', 16), int('35', 16), int('85', 16), int(

        'e2', 16), int('f9', 16), int('37', 16), int('e8', 16), int('1c', 16), int('75', 16), int('df', 16), int('6e', 16)],

    [int('47', 16), int('f1', 16), int('1a', 16), int('71', 16), int('1d', 16), int('29', 16), int('c5', 16), int('89', 16), int(

        '6f', 16), int('b7', 16), int('62', 16), int('0e', 16), int('aa', 16), int('18', 16), int('be', 16), int('1b', 16)],

    [int('fc', 16), int('56', 16), int('3e', 16), int('4b', 16), int('c6', 16), int('d2', 16), int('79', 16), int('20', 16), int(

        '9a', 16), int('db', 16), int('c0', 16), int('fe', 16), int('78', 16), int('cd', 16), int('5a', 16), int('f4', 16)],

    [int('1f', 16), int('dd', 16), int('a8', 16), int('33', 16), int('88', 16), int('07', 16), int('c7', 16), int('31', 16), int(

        'b1', 16), int('12', 16), int('10', 16), int('59', 16), int('27', 16), int('80', 16), int('ec', 16), int('5f', 16)],

    [int('60', 16), int('51', 16), int('7f', 16), int('a9', 16), int('19', 16), int('b5', 16), int('4a', 16), int('0d', 16), int(

        '2d', 16), int('e5', 16), int('7a', 16), int('9f', 16), int('93', 16), int('c9', 16), int('9c', 16), int('ef', 16)],

    [int('a0', 16), int('e0', 16), int('3b', 16), int('4d', 16), int('ae', 16), int('2a', 16), int('f5', 16), int('b0', 16), int(

        'c8', 16), int('eb', 16), int('bb', 16), int('3c', 16), int('83', 16), int('53', 16), int('99', 16), int('61', 16)],

    [int('17', 16), int('2b', 16), int('04', 16), int('7e', 16), int('ba', 16), int('77', 16), int('d6', 16), int('26', 16), int(

        'e1', 16), int('69', 16), int('14', 16), int('63', 16), int('55', 16), int('21', 16), int('0c', 16), int('7d', 16)]

]

def lookup(byte):

    x = byte >> 4

    y = byte & 15

    return aes\_sbox[x][y]

def reverse\_lookup(byte):

    x = byte >> 4

    y = byte & 15

    return reverse\_aes\_sbox[x][y]

def break\_in\_grids\_of\_16(s):

    all = []

    for i in range(len(s)//16):

        b = s[i\*16: i\*16 + 16]

        grid = [[], [], [], []]

        for i in range(4):

            for j in range(4):

                grid[i].append(b[i + j\*4])

        all.append(grid)

    return all

def expand\_key(key, rounds):

    rcon = [[1, 0, 0, 0]]

    for \_ in range(1, rounds):

        rcon.append([rcon[-1][0]\*2, 0, 0, 0])

        if rcon[-1][0] > 0x80:

            rcon[-1][0] ^= 0x11b

    key\_grid = break\_in\_grids\_of\_16(key)[0]

    for round in range(rounds):

        last\_column = [row[-1] for row in key\_grid]

        last\_column\_rotate\_step = rotate\_row\_left(last\_column)

        last\_column\_sbox\_step = [lookup(b) for b in last\_column\_rotate\_step]

        last\_column\_rcon\_step = [last\_column\_sbox\_step[i]

                                ^ rcon[round][i] for i in range(len(last\_column\_rotate\_step))]

        for r in range(4):

            key\_grid[r] += bytes([last\_column\_rcon\_step[r]

                                ^ key\_grid[r][round\*4]])

        # Three more columns to go

        for i in range(len(key\_grid)):

            for j in range(1, 4):

                key\_grid[i] += bytes([key\_grid[i][round\*4+j]

                                    ^ key\_grid[i][round\*4+j+3]])

    return key\_grid

def rotate\_row\_left(row, n=1):

    return row[n:] + row[:n]

def multiply\_by\_2(v):

    s = v << 1

    s &= 0xff

    if (v & 128) != 0:

        s = s ^ 0x1b

    return s

def multiply\_by\_3(v):

    return multiply\_by\_2(v) ^ v

def mix\_columns(grid):

    new\_grid = [[], [], [], []]

    for i in range(4):

        col = [grid[j][i] for j in range(4)]

        col = mix\_column(col)

        for i in range(4):

            new\_grid[i].append(col[i])

    return new\_grid

def mix\_column(column):

    r = [

        multiply\_by\_2(column[0]) ^ multiply\_by\_3(

            column[1]) ^ column[2] ^ column[3],

        multiply\_by\_2(column[1]) ^ multiply\_by\_3(

            column[2]) ^ column[3] ^ column[0],

        multiply\_by\_2(column[2]) ^ multiply\_by\_3(

            column[3]) ^ column[0] ^ column[1],

        multiply\_by\_2(column[3]) ^ multiply\_by\_3(

            column[0]) ^ column[1] ^ column[2],

    ]

    return r

def add\_sub\_key(block\_grid, key\_grid):

    r = []

    # 4 rows in the grid

    for i in range(4):

        r.append([])

        # 4 values on each row

        for j in range(4):

            r[-1].append(block\_grid[i][j] ^ key\_grid[i][j])

    return r

def extract\_key\_for\_round(expanded\_key, round):

    return [row[round\*4: round\*4 + 4] for row in expanded\_key]

def enc(key, data):

    # First we need to padd the data with \x00 and break it into blocks of 16

    pad = bytes(16 - len(data) % 16)

    if len(pad) != 16:

        data += pad

    grids = break\_in\_grids\_of\_16(data)

    # Now we need to expand the key for the multiple rounds

    expanded\_key = expand\_key(key, 11)

    # And apply the original key to the blocks before start the rounds

    # For now on we will work with integers

    temp\_grids = []

    round\_key = extract\_key\_for\_round(expanded\_key, 0)

    for grid in grids:

        temp\_grids.append(add\_sub\_key(grid, round\_key))

    grids = temp\_grids

    # Now we can move to the main part of the algorithm

    for round in range(1, 10):

        temp\_grids = []

        for grid in grids:

            sub\_bytes\_step = [[lookup(val) for val in row] for row in grid]

            shift\_rows\_step = [rotate\_row\_left(

                sub\_bytes\_step[i], i) for i in range(4)]

            mix\_column\_step = mix\_columns(shift\_rows\_step)

            round\_key = extract\_key\_for\_round(expanded\_key, round)

            add\_sub\_key\_step = add\_sub\_key(mix\_column\_step, round\_key)

            temp\_grids.append(add\_sub\_key\_step)

        grids = temp\_grids

    # A final round without the mix columns

    temp\_grids = []

    round\_key = extract\_key\_for\_round(expanded\_key, 10)

    for grid in grids:

        sub\_bytes\_step = [[lookup(val) for val in row] for row in grid]

        shift\_rows\_step = [rotate\_row\_left(

            sub\_bytes\_step[i], i) for i in range(4)]

        add\_sub\_key\_step = add\_sub\_key(shift\_rows\_step, round\_key)

        temp\_grids.append(add\_sub\_key\_step)

    grids = temp\_grids

    # Just need to recriate the data into a single stream before returning

    int\_stream = []

    for grid in grids:

        for column in range(4):

            for row in range(4):

                int\_stream.append(grid[row][column])

    return bytes(int\_stream)

def dec(key, data):

    grids = break\_in\_grids\_of\_16(data)

    expanded\_key = expand\_key(key, 11)

    temp\_grids = []

    round\_key = extract\_key\_for\_round(expanded\_key, 10)

    # First we undo the final round

    temp\_grids = []

    for grid in grids:

        add\_sub\_key\_step = add\_sub\_key(grid, round\_key)

        shift\_rows\_step = [rotate\_row\_left(

            add\_sub\_key\_step[i], -1 \* i) for i in range(4)]

        sub\_bytes\_step = [[reverse\_lookup(val) for val in row] for row in shift\_rows\_step]

        temp\_grids.append(sub\_bytes\_step)

    grids = temp\_grids

    for round in range(9, 0, -1):

        temp\_grids = []

        for grid in grids:

            round\_key = extract\_key\_for\_round(expanded\_key, round)

            add\_sub\_key\_step = add\_sub\_key(grid, round\_key)

            # Doing the mix columns three times is equal to using the reverse matrix

            mix\_column\_step = mix\_columns(add\_sub\_key\_step)

            mix\_column\_step = mix\_columns(mix\_column\_step)

            mix\_column\_step = mix\_columns(mix\_column\_step)

            shift\_rows\_step = [rotate\_row\_left(

                mix\_column\_step[i], -1 \* i) for i in range(4)]

            sub\_bytes\_step = [

                [reverse\_lookup(val) for val in row] for row in shift\_rows\_step]

            temp\_grids.append(sub\_bytes\_step)

        grids = temp\_grids

        temp\_grids = []

    # Reversing the first add sub key

    round\_key = extract\_key\_for\_round(expanded\_key, 0)

    for grid in grids:

        temp\_grids.append(add\_sub\_key(grid, round\_key))

    grids = temp\_grids

    # Just transform the grids back to bytes

    int\_stream = []

    for grid in grids:

        for column in range(4):

            for row in range(4):

                int\_stream.append(grid[row][column])

    return bytes(int\_stream)

def test\_aes():

    key = bytes.fromhex('2b7e151628aed2a6abf7158809cf4f3c')

    plaintext = bytes.fromhex('3243f6a8885a308d313198a2e0370734')

    print("Plaintext: ",key)

    ciphertext = enc(key, plaintext)

    decrypted\_plaintext = dec(key, ciphertext)

    print("Ciphertext: ",ciphertext)

    assert plaintext == decrypted\_plaintext

    print("Decrypted: ", decrypted\_plaintext)

    print("AES encryption and decryption test passed successfully.")

# Run the test

test\_aes()

**OUTPUT:**

Plaintext: b'+~\x15\x16(\xae\xd2\xa6\xab\xf7\x15\x88\t\xcfO<'

Ciphertext: b'9%\x84\x1d\x02\xdc\t\xfb\xdc\x11\x85\x97\x19j\x0b2'

Decrypted: b'2C\xf6\xa8\x88Z0\x8d11\x98\xa2\xe07\x074'

AES encryption and decryption test passed successfully.