```
import sys

# S-Box
sBox  = [0x9, 0x4, 0xa, 0xb, 0xd, 0x1, 0x8, 0x5,
         0x6, 0x2, 0x0, 0x3, 0xc, 0xe, 0xf, 0x7]

# Inverse S-Box
sBoxI = [0xa, 0x5, 0x9, 0xb, 0x1, 0x7, 0x8, 0xf,
         0x6, 0x0, 0x2, 0x3, 0xc, 0x4, 0xd, 0xe]

# Round keys: K0 = w0 + w1; K1 = w2 + w3; K2 = w4 + w5
w = [None] * 6


def mult(p1, p2):
    #Multiply two polynomials in (GF)(2^4)/x^4 + x + 1
    p = 0
    while p2:
        if p2 & 0b1:
            p ^= p1
        p1 <<= 1
        if p1 & 0b10000:
            p1 ^= 0b11
        p2 >>= 1
    return p & 0b1111


def intToVec(n):
    #Convert a 2-byte integer into a 4-element vector
    return [n >> 12, (n >> 4) & 0xf, (n >> 8) & 0xf,  n & 0xf]

def vecToInt(m):
    #Convert a 4-element vector into 2-byte integer
    return (m[0] << 12) + (m[2] << 8) + (m[1] << 4) + m[3]


def addKey(s1, s2):
    #Add two keys in GF(2^4)
    return [i ^ j for i, j in zip(s1, s2)]

def nibble_substitution(sbox,s):
    return [sbox[e] for e in s]

def shiftRow(s):
    return [s[0], s[1], s[3], s[2]]


def sub2Nib(b):
    #Swap each nibble and substitute it using sBox
    #accepts 8 bit key, thus taken nibble by nibble
    return sBox[b >> 4] + (sBox[b & 0x0f] << 4)

def keyExp(key):
    Rcon1, Rcon2 = 0b10000000, 0b00110000
    w[0] = (key & 0xff00) >> 8
    w[1] = key & 0x00ff
    w[2] = w[0] ^ Rcon1 ^ sub2Nib(w[1])
    w[3] = w[2] ^ w[1]
    w[4] = w[2] ^ Rcon2 ^ sub2Nib(w[3])
    w[5] = w[4] ^ w[3]


def get_key(w):
    keys = [((w[0] << 8) + w[1]), ((w[2] << 8) + w[3]), ((w[4] << 8) + w[5])]
    return keys


def mixCol(s):
    return [s[0] ^ mult(4, s[2]), s[1] ^ mult(4, s[3]),
    s[2] ^ mult(4, s[0]), s[3] ^ mult(4, s[1])]


def encrypt(plain_text):

    keys = get_key(w)

    state = intToVec(keys[0] ^ plain_text)

    # Round 1
    state = nibble_substitution(sBox,state)
    state = shiftRow(state)
    state = mixCol(state)

    state = addKey(intToVec(keys[1]), state)

    # Round 2
    state = nibble_substitution(sBox,state)
    state = shiftRow(state)

    state = addKey(intToVec(keys[2]), state)

    return vecToInt(state)


def iMixCol(s):
    return [mult(9, s[0]) ^ mult(2, s[2]), mult(9, s[1]) ^ mult(2, s[3]),
        mult(9, s[2]) ^ mult(2, s[0]), mult(9, s[3]) ^ mult(2, s[1])]


def decrypt(ctext):
    state = intToVec(((w[4] << 8) + w[5]) ^ ctext)
    state = nibble_substitution(sBoxI, shiftRow(state))
    state = iMixCol(addKey(intToVec((w[2] << 8) + w[3]), state))
    state = nibble_substitution(sBoxI, shiftRow(state))

    return vecToInt(addKey(intToVec((w[0] << 8) + w[1]), state))


if __name__ == '__main__':

    plaintext = int(input("Enter plaintext (Numeric value < 65536): "))
    key = int(input("Enter key (Numeric value): "))
```

```
keyExp(key)

ciphertext = encrypt(plaintext)
print("Encrypted text: ", ciphertext)

dec = decrypt(ciphertext)

print("Decrypted text: ", dec)
```

```
Enter plaintext (Numeric value < 65536): 450
Enter key (Numeric value): 800
Encrypted text:  33439
Decrypted text:  450
```