```
import numpy as np
```

Utility functions

Key Gen Functions

```
def rot_word(w):
  return [w[1], w[0]]

def sub_word(w):
  S = [
       ['9', '4', 'a', 'b'],
       ['d', '1', '8', '5'],
       ['6', '2', '0', '3'],
       ['c', 'e', 'f', '7']
  ]
  w_new = []
  for h in w:
    n = hex_to_nibble(h)
    n_new = S[int(n[:2], 2)][int(n[2:], 2)]
    w_new.append(n_new)
  return w_new

def xor(w1, w2):
  w = []
  for i in range(2):
    x = int(hex_to_nibble(w1[i]), 2)
    y = int(hex_to_nibble(w2[i]), 2)
    w.append(nibble_to_hex(bin(x^y)[2:]))
  return w

def key_expansion(k):
  w0, w1 = k[:2], k[2:]
  r1 = ['8', '0']
  t2 = xor(sub_word(rot_word(w1)), r1)
  w2 = xor(w0, t2)
  w3 = xor(w1, w2)
  r2 = ['3', '0']
  t4 = xor(sub_word(rot_word(w3)), r2)
  w4 = xor(w2, t4)
  w5 = xor(w3, w4)
  return w0 + w1, w2 + w3, w4 + w5
```

Conversion functions

```
def nibble_to_hex(n):
  assert len(n) <= 4, 'Invalid nibble provided.'
  if len(n) < 4: n = (4-len(n))*'0' + n
  return hex(int(n, 2))[2:]
```

```
def hex_to_nibble(h):
  assert len(h) == 1, 'Invalid hex digit.'
  n = bin(int(h,16))[2:]
  return (4-len(n))*'0' + n
```

```
def block_to_state(b):
  return [
          [b[0], b[2]],
          [b[1], b[3]]
  ]
```

```
def state_to_block(s):
  return [s[0][0], s[1][0], s[0][1], s[1][1]]
```

```
def sub_nibbles(s):
  S = [
       ['9', '4', 'a', 'b'],
       ['d', '1', '8', '5'],
       ['6', '2', '0', '3'],
       ['c', 'e', 'f', '7']
  ]
  b = state_to_block(s)
  b_new = []
  for h in b:
    n = hex_to_nibble(h)
    n_new = S[int(n[:2], 2)][int(n[2:], 2)]
    b_new.append(n_new)
  return block_to_state(b_new)
```

```
def shift_rows(s):
  return [
          [s[0][0], s[0][1]],
          [s[1][1], s[1][0]]
  ]
```

```
def mul(x, y):
  p1 = [int(c) for c in hex_to_nibble(x)]
  p2 = [int(c) for c in hex_to_nibble(y)]
  return np.polymul(p1, p2)

def add(x, y):
  p = list(np.polyadd(x, y))
  p = [c%2 for c in p]
  _, r = np.polydiv(p, [1, 0, 0, 1, 1])
  r = [str(int(c%2)) for c in r]
  return nibble_to_hex(''.join(r))
```

```
12
13 def mix_columns(s):
14   C = [
15        ['1', '4'],
16        ['4', '1']
17   ]
18   s_new = [
19            [None, None],
20            [None, None]
21   ]
22   for i in range(2):
23     for j in range(2):
24       s_new[i][j] = add(mul(C[i][0], s[0][j]), mul(C[i][1], s[1][j]))
25   return s_new
```

```
1 def add_round_key(k, s):
2   k_state = block_to_state(k)
3   w1 = xor([k_state[0][0], k_state[1][0]], [s[0][0], s[1][0]])
4   w2 = xor([k_state[0][1], k_state[1][1]], [s[0][1], s[1][1]])
5   return [
6           [w1[0], w2[0]],
7           [w1[1], w2[1]]
8   ]
```

```
1 def bin_to_dec(x):
2   return int(x, 2)
3 def dec_to_bin(x):
4   return bin(x).replace("0b","")
5 def hex_to_bin(x):
6   ret = dec_to_bin(int(x, 16))
7   ret = assert_value_size(ret, len(x)*4)
8   return ret
9 def bin_to_hex(x):
10  return hex(bin_to_dec(x))
```

```
1 def assert_value_size(x, s):
2   while len(x) < s:
3     x = "0" + x
4   return x
```

```
1 def xor(a, b):
2   ret = ""
3   for i in range(len(a)):
4     if a[i] == b[i]: ret += "0"
5     else: ret += "1"
6   return ret
```

```
1 def split_str(val):
2   half = len(val)//2
3   return val[:half], val[half:]
```

```
1 def get_indices(nib):
```

```
def get_indices(nib):
  r = bin_to_dec(nib[:2])
  c = bin_to_dec(nib[2:])
  return r, c
```

```
def nibble_list(x):
  x = assert_value_size(x, 16)
  ret = [x[i:i+4] for i in range(0, len(x), 4)]
  return ret

def list_to_mat(l):
  return [
      [l[0], l[2]],
      [l[1], l[3]]
  ]

def mat_to_list(m):
  return [m[0][0], m[1][0], m[0][1], m[1][1]]
```

```
def rot_nib(val):
  half = len(val)//2
  return val[half:] + val[:half]
```

```
def mul_nib(nib1, nib2):
  p1 = [int(c) for c in nib1]
  p2 = [int(c) for c in nib2]
  ret = np.polymul(p1, p2)
  ret = [str(c) for c in ret]
  return "".join(ret)

def add_nib(nib1, nib2):
  p1 = [int(c) for c in nib1]
  p2 = [int(c) for c in nib2]
  ret = np.polyadd(p1, p2)
  ret = [c % 2 for c in ret]
  _, r = np.polydiv(ret, [1, 0, 0, 1, 1])
  nib = [str(int(c%2)) for c in r]
  nib = "".join(nib)
  while len(nib) > 4:
    nib = nib[1:]
  nib = assert_value_size(nib, 4)
  return nib
```

```

```

```
def gen_inv_s_box(s):
  ret = [r[:] for r in s]
  for i in range(4):
    for j in range(4):
      r, c = get_indices(hex_to_bin(s[i][j]))
      ret[r][c] = bin_to_hex(assert_value_size(dec_to_bin(i), 2) + assert_value_
  return ret
```

```
S = [
    ["1", "2", "3", "4"],
     ["5", "6", "7", "8"],
     ["9", "A", "B", "C"],
     ["D", "E", "F", "0"]
]
INV_S = gen_inv_s_box(S)
M = [
     ["1", "4"],
     ["4", "1"]
]
INV_M = [
    ["9", "2"],
    ["2", "9"]
]
print(INV_S)
```

```
[['f', '0', '1', '2'], ['3', '4', '5', '6'], ['7', '8', '9', 'a'], ['b', 'c',
```

```
def sub_nib(x, s):
  ret = ""
  for i in range(0, len(x), 4):
    nib = x[i:i+4]
    r, c = get_indices(nib)
    ret += hex_to_bin(s[r][c])
  return ret

def sub_nibs(x, s):
  for i in range(len(x)):
    for j in range(len(x[i])):
      x[i][j] = sub_nib(x[i][j], s)
  return x
```

```
def mixcol(A, B):
  ret = [
      [None, None],
      [None, None]
  ]
  for i in [0, 1]:
    for j in [0, 1]:
      ret[i][j] = add_nib(mul_nib(A[i][0], B[0][j]), mul_nib(A[i][1], B[1][j]))
  return ret
```

```
def shift_row(state):
  state[1][0], state[1][1] = state[1][1], state[1][0]
  return state
```

```
def add_round_key(state, key):
  k_mat = list_to_mat(nibble_list(key))
  for i in range(2):
    for j in range(2):
```

```
      state[i][j] = xor(state[i][j], k_mat[i][j])
  return state
```

Key Generation

```
def get_subkey(prev_key, t):
  w0, w1 = split_str(prev_key)
  w2 = w0
  w2 = xor(w2, t)
  w2 = xor(w2, sub_nib(rot_nib(w1), S))
  w3 = xor(w2, w1)
  print(w2,w3)
  return w2 + w3
```

```
def gen_subkeys(key):
  key0 = key
  key1 = get_subkey(key0, hex_to_bin("80"))
  key2 = get_subkey(key1, hex_to_bin("60"))
  return key0, key1, key2
```

Encryption

```
def encrypt(plaintext, key):
  key0, key1, key2 = gen_subkeys(key)
  print("Keys:", key0, key1, key2)
  state = list_to_mat(nibble_list(plaintext))

  # Round 0
  state = add_round_key(state, key0)

  #Round 1
  state = sub_nibs(state, S)
  state = shift_row(state)
  state = mixcol(M, state)
  state = add_round_key(state, key1)

  # Round 2
  state = sub_nibs(state, S)
  state = shift_row(state)
  state = add_round_key(state, key2)


  ciphertext = "".join(mat_to_list(state))

  return ciphertext
```

Decryption

```
def decrypt(ciphertext, key):
  key0, key1, key2 = gen_subkeys(key)
  state = list_to_mat(nibble_list(ciphertext))

  # Inv round 2
  state = add_round_key(state, key2)
  state = shift_row(state)
  state = sub_nibs(state, INV_S)

  # Inv round 1
  state = add_round_key(state, key1)
  state = mixcol(INV_M, state)
  state = shift_row(state)
  state = sub_nibs(state, INV_S)

  # Inv round 0
  state = add_round_key(state, key0)

  plaintext = "".join(mat_to_list(state))
  return plaintext
```

Testing

```
plaintext = hex_to_bin("BC78")
key = hex_to_bin("2B85")
```

```
print('Plain Text:', plaintext)
print('Key:', key)
```

```
Plain Text: 1011110001111000
Key: 0010101110000101
```

```
c = encrypt(plaintext, key)
p = decrypt(c, key)
```

```
11000010 01000111
00100111 01100000
Keys: 0010101110000101 1100001001000111 0010011101100000
11000010 01000111
00100111 01100000
```

```
print('Encrypted Text:', c)
print('Decrypted Text:', p)
```

```
Encrypted Text: 0101011100111101
```

```
Decrypted Text: 1011110001111000
```

```
1 assert(p == plaintext)
```

✓ 0s completed at 09:55