In this project, I compared the efficiency of encryption and decryption between the DES algorithm and the AES algorithm.

1. I studied the implementation of the AES algorithm and implemented it myself. There are some challenges but i fix them, i also change some basic logic to apply the algorithm easily for myself.
2. I integrated the algorithm based on existing encryption libraries. You need `pip install pycryptodome`
3. Following that, I designed a file for timing tests and another file for generating random data to run the tests. You can get encrypt and decrypt time by custom DES, AES and library DES, AES.
4. Ultimately, I implemented a GUI interface for the encryption process, allowing for a visual comparison of the performance between different algorithms.

AES realization

AES called Advanced Encryption Standard, is a safer encrypt algorithm compared by DES, and also has better performance. It divides the plaintext into some 16 bytes(128 bits) blocks, and the key needs 128, 192 or 256 bits length, the difference of them needs different rounds. And a byte is 2 hex data, so 16 bytes can be 4 * 4 matrix.

> In the official lectures, the sorting is distinguished by columns, which means that the data is distributed from top to bottom and then from left to right, but the disadvantage of this is that the representation of the matrix is not easy, because it is easy to deal with each sub-matrix in a two-position matrix, if you want to deal with each sub-matrix in a certain position, it may require more time complexity. So I changed the column sort to row sort, and made some adjustments to the following steps.

```
'''
    [0,  4,  8,  12],        [0,  1,  2,  3 ],
    [1,  5,  9,  13],   ==>  [4,  5,  6,  7 ],
    [2,  6,  10, 14],        [8,  9,  10, 11],
    [3,  7,  11, 15]         [12, 13, 14, 15]
'''
```

Fence 1

1. The Encrypt flow

1.1 Initial transform

The initial byte matrix requires an XOR operation with the initial key, and then a new state matrix is obtained

1.2 Loop operation

128 bits key will get 9 times, 192 bits key will get 11 times, 256 bits will get 13 times. In this project, all condition can be accomplished, you just need to type 4, 6, 8 to change the key length. 4 is for 128 bits, 6 is for 192 bits, 8 is for 256 bits

1.2.1 SubBytes

Use the fix S-Box, this is a 16*16 size matrix, so the 16 hex data will show the position of the opposite value, the original value will be replaced. Every value in state matrix will be changed.

```
s_box = [
    [0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b,
0xfe, 0xd7, 0xab, 0x76],
    [0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf,
0x9c, 0xa4, 0x72, 0xc0],
    [0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1,
0x71, 0xd8, 0x31, 0x15],
    [0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2,
0xeb, 0x27, 0xb2, 0x75],
    [0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3,
0x29, 0xe3, 0x2f, 0x84],
    [0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39,
0x4a, 0x4c, 0x58, 0xcf],
    [0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f,
0x50, 0x3c, 0x9f, 0xa8],
    [0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21,
0x10, 0xff, 0xf3, 0xd2],
    [0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d,
0x64, 0x5d, 0x19, 0x73],
    [0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14,
0xde, 0x5e, 0x0b, 0xdb],
    [0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62,
0x91, 0x95, 0xe4, 0x79],
    [0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea,
0x65, 0x7a, 0xae, 0x08],
    [0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f,
0x4b, 0xbd, 0x8b, 0x8a],
    [0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9,
0x86, 0xc1, 0x1d, 0x9e],
    [0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9,
0xce, 0x55, 0x28, 0xdf],
    [0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f,
0xb0, 0x54, 0xbb, 0x16]
]
# 找到s盒中的值
def find_s(num):
    return s_box[num // 16][num % 16]

def SubBytes(state):
    for i in range(4):
        for j in range(4):
            state[i][j] = find_s(state[i][j])
    return state
```

Fence 2

1.2.2 ShiftRows

In this function, every row in the state matrix will shift left, the first row will be stable, the second row will shift 1 position, the 3rd row will shift 2 position, the 4th row will shift 3 position. However, i use the row ordering matrix, so in my function, the first col will stable, the 2nd col will shift top 1 position, 3rd shift top 2 position, 4th shift top 3 position.

```
'''old ordering state
[[0, 4, 10, 12],
 [1, 5, 11, 13],
 [2, 6, 14, 15],
 [3, 7, 12, 14]]
after shift
[[0, 4, 10, 12],
 [5, 11, 13, 1],
 [14, 15, 2, 6],
 [14, 3, 7, 12]]

current ordernning state
[[0, 1, 2, 3],
 [4, 5, 6, 7],
 [8, 9, 10, 11],
 [12, 13, 14, 15]]
after shift
[[0, 5, 10, 15],
 [4, 9, 14, 3],
 [8, 13, 2, 7],
 [12, 1, 6, 11]]'''
# 行移位
def ShiftRows(state):
    return [
        [state[0][0], state[1][1], state[2][2], state[3][3]],  # 对角线1
        [state[1][0], state[2][1], state[3][2], state[0][3]],  # 对角线偏移1
        [state[2][0], state[3][1], state[0][2], state[1][3]],  # 对角线偏移2
        [state[3][0], state[0][1], state[1][2], state[2][3]]   # 对角线偏移3
    ]
```

Fence 3

1.2.3 MixColumns

This is a difficult function to apply, you need learn a new math knowledge, GF(2^8). This is a galois field mind, which means the limitation, the result should be in 0-255, so the add function will be XOR, the multiple function will use a special method.

In official doc, every column will be calculated, however, in my mind, every row will be calculated. The operation will follow below steps.

```
def xtime(num):
    shift_num = num << 1
    if num & 0x80:
        shift_num = shift_num ^ 0x1b
    return shift_num
def mul_GF(a, b):
    result = 0
    for _ in range(8):
```

```
        if b & 0x01:
            result ^= a
        a = xtime(a)
        b >>= 1
    return result & 0xFF
def MixColumns(state):
    for i in range(4):
        row = [state[i][j] for j in range(4)]
        state[i][0] = mul_GF(0x02, row[0]) ^ mul_GF(0x03, row[1]) ^
mul_GF(0x01, row[2]) ^ mul_GF(0x01, row[3])
        state[i][1] = mul_GF(0x01, row[0]) ^ mul_GF(0x02, row[1]) ^
mul_GF(0x03, row[2]) ^ mul_GF(0x01, row[3])
        state[i][2] = mul_GF(0x01, row[0]) ^ mul_GF(0x01, row[1]) ^
mul_GF(0x02, row[2]) ^ mul_GF(0x03, row[3])
        state[i][3] = mul_GF(0x03, row[0]) ^ mul_GF(0x01, row[1]) ^
mul_GF(0x01, row[2]) ^ mul_GF(0x02, row[3])
    return state
```

Fence 4

1.2.4 AddRoundKey

This is a step to make state matrix XOR with sub-keys.

```
# round will show the loop condition, every 4 rows is a sub-key
def AddRoundKey(state, keys, round):
    for i in range(4):
        for j in range(4):
            state[i][j] ^= keys[round * 4 + i][j]
    return state
```

Fence 5

1.3 Final operation

Just use `SubBytes, ShiftRows, AddRoundKey`

1.4 Whole method

```
def AES_encrypt(plaintext, key, condition=4):
    keys = generate_keys(key, condition)
    padded_plaintext = pad(plaintext)
    ciphertext = ''
    for i in range(len(padded_plaintext) // 32):
        block = padded_plaintext[i * 32:(i + 1) * 32]
        state = bytes2matrix(block)
        state = AddRoundKey(state, keys, 0)
        for i in range(1, condition + 7):
            state = SubBytes(state)
            state = ShiftRows(state)
            if i != condition + 6:
                state = MixColumns(state)
            state = AddRoundKey(state, keys, i)
        ciphertext += matrix2bytes(state)
    return ciphertext
```

Fence 6

2. Generate sub-keys

You need use the original key to generate many sub keys, every key is 16 bytes, every 4 bytes will be one row

```
Rcon = [0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1B, 0x36, 0x6C, 0xD8,
0xAB, 0x4D]
def SubWord(list):
    return [find_s(list[j]) for j in range(4)]
def RotWord(list):
    return list[1:] + list[:1]
def T_function(list, i):
    list = RotWord(list)
    list = SubWord(list)
    list[0] = list[0] ^ Rcon[i]
    return list
# 4 is for 128 bits, 6 is for 192 bits, 8 is for 256 bits
def generate_keys(key, condition=4):
    k_hex = pbkdf2_hmac("sha256", key.encode("utf-8"), b"salt", 10000,
dklen=condition * 4)
    rounds = condition + 6
    k_hex_list = [int(f"{b:02x}", 16) for b in k_hex]
    keys = [k_hex_list[i:i+4] for i in range(0, len(k_hex_list), 4)]
    for i in range(condition, 4 * (rounds + 1)):
        temp = keys[i-1]
        if i % condition == 0:
            temp = T_function(temp, i // condition - 1)
        elif condition > 6 and i % condition == 4:
            temp = SubWord(temp)
        keys.append([keys[i-condition][j] ^ temp[j] for j in range(4)])
    return keys
```

Fence 7

3. The Decrypt flow

It is very similar with encrypt flow, but every operation will be invrese. Like ShiftRows, shift down position. Use inverse S-box, and in the decrypt function, the steps will be updated.

```
inv_s_box = [
    [0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e,
0x81, 0xf3, 0xd7, 0xfb],
    [0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44,
0xc4, 0xde, 0xe9, 0xcb],
    [0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b,
0x42, 0xfa, 0xc3, 0x4e],
    [0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49,
0x6d, 0x8b, 0xd1, 0x25],
    [0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc,
0x5d, 0x65, 0xb6, 0x92],
    [0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57,
0xa7, 0x8d, 0x9d, 0x84],
    [0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05,
0xb8, 0xb3, 0x45, 0x06],
```

```
    [0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03,
0x01, 0x13, 0x8a, 0x6b],
    [0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce,
0xf0, 0xb4, 0xe6, 0x73],
    [0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8,
0x1c, 0x75, 0xdf, 0x6e],
    [0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e,
0xaa, 0x18, 0xbe, 0x1b],
    [0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe,
0x78, 0xcd, 0x5a, 0xf4],
    [0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59,
0x27, 0x80, 0xec, 0x5f],
    [0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f,
0x93, 0xc9, 0x9c, 0xef],
    [0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c,
0x83, 0x53, 0x99, 0x61],
    [0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63,
0x55, 0x21, 0x0c, 0x7d],
]
def inv_find_s(num):
    return inv_s_box[num // 16][num % 16]

def inv_SubBytes(state):
    for i in range(4):
        for j in range(4):
            state[i][j] = inv_find_s(state[i][j])
    return state

def inv_ShiftRows(state):
    return [
        [state[0][0], state[3][1], state[2][2], state[1][3]],
        [state[1][0], state[0][1], state[3][2], state[2][3]],
        [state[2][0], state[1][1], state[0][2], state[3][3]],
        [state[3][0], state[2][1], state[1][2], state[0][3]]
    ]

def inv_MixColumns(state):
    for i in range(4):
        row = [state[i][j] for j in range(4)]
        state[i][0] = mul_GF(0x0e, row[0]) ^ mul_GF(0x0b, row[1]) ^
mul_GF(0x0d, row[2]) ^ mul_GF(0x09, row[3])
        state[i][1] = mul_GF(0x09, row[0]) ^ mul_GF(0x0e, row[1]) ^
mul_GF(0x0b, row[2]) ^ mul_GF(0x0d, row[3])
        state[i][2] = mul_GF(0x0d, row[0]) ^ mul_GF(0x09, row[1]) ^
mul_GF(0x0e, row[2]) ^ mul_GF(0x0b, row[3])
        state[i][3] = mul_GF(0x0b, row[0]) ^ mul_GF(0x0d, row[1]) ^
mul_GF(0x09, row[2]) ^ mul_GF(0x0e, row[3])
    return state


def AES_decrypt(ciphertext, key, condition=4):
    keys = generate_keys(key, condition)
    padded_ciphertext = ciphertext
    plaintext = ''
    for i in range(len(padded_ciphertext) // 32):
        block = padded_ciphertext[i * 32:(i + 1) * 32]
```

```
        state = bytes2matrix(block)
        state = AddRoundKey(state, keys, condition + 6)
        for i in range(condition + 5, -1, -1):
            state = inv_ShiftRows(state)
            state = inv_SubBytes(state)
            state = AddRoundKey(state, keys, i)
            if i != 0:
                state = inv_MixColumns(state)
        plaintext += matrix2bytes(state)
    plaintext = unpad(plaintext)
    return hex2str(plaintext)
```

Fence 8

4. Library DES & AES

```
from Crypto.Cipher import AES, DES
from hashlib import pbkdf2_hmac


def get_des_key(key):
    return pbkdf2_hmac("sha256", key.encode("utf-8"), b"salt", 10000, dklen=8)

def get_aes_key(key):
    return pbkdf2_hmac("sha256", key.encode("utf-8"), b"salt", 10000, dklen=16)

def des_encrypt(data, key):
    key = get_des_key(key)
    data_bytes = data.encode()
    if len(data_bytes) % 8 != 0:
            padding_length = 8 - (len(data_bytes) % 8)
            data_bytes += bytes([padding_length] * padding_length)
    cipher = DES.new(key, DES.MODE_ECB)
    encrypted = cipher.encrypt(data_bytes)
    return encrypted

def des_decrypt(encrypted_data, key):
    key = get_des_key(key)
    cipher = DES.new(key, DES.MODE_ECB)
    decrypted = cipher.decrypt(encrypted_data)
    padding_length = decrypted[-1]
    return decrypted[:-padding_length]


def aes_encrypt(data, key):
    key = get_aes_key(key)
    data_bytes = data.encode()
    if len(data_bytes) % 16 != 0:
        padding_length = 16 - (len(data_bytes) % 16)
        data_bytes += bytes([padding_length] * padding_length)
    cipher = AES.new(key, AES.MODE_ECB)
    encrypted = cipher.encrypt(data_bytes)
    return encrypted
```

```
def aes_decrypt(encrypted_data, key):
    key = get_aes_key(key)
    cipher = AES.new(key, AES.MODE_ECB)
    decrypted = cipher.decrypt(encrypted_data)
    padding_length = decrypted[-1]
    return decrypted[:-padding_length]
```

Fence 9

5. Time compare basic logic

Just get the time stamp before one operation, then get the time stamp after one operation, minus these two time will get the cost time.

```
def get_time_cost(func, *args):
    start_time = time.perf_counter()
    result = func(*args)
    end_time = time.perf_counter()
    time_cost = end_time - start_time
    return result, time_cost
```

Fence 10