# Lab 7 Report Template

**Given a string S and a key string P, encode and decode the string S, using the key string P .**

## 1. P as integer, to XOR every charactor to encode and decode

1.1 In fact, when I begin to use some mathematic methods to encode a character, I need to change it into a number by ASCii or something else. However, in this question, the string P is an integral string, which means that step is not necessary.

1.2 I get the character form P and S in turn and I encode them by using ASCii, then they will be used in the calculation:

(Pitem XOR Sitem). Then I decode the result with ASCii.

1.3 In the decode function, the operations are same to the encode function.

1.4 Code in Python :

```
def xor_encrypt(tips,key):
    lkey = len(key)
    secret = []
    num = 0
    for each in tips:
        if num >= lkey:
            num = num % lkey
        secret.append(chr(ord(each) ^ ord(key[num])))
        num += 1

    #return b64.b64encode( "".join( secret ).encode() ).decode()
    return "".join(secret)
```

```python
def xor_decrypt(secret,key):
    #tips = b64.b64decode( secret.encode() ).decode()
    tips = secret
    lkey = len(key)
    secret = []
    num = 0
    for each in tips:
        if num >= lkey:
            num = num % lkey
        secret.append(chr(ord(each) ^ ord(key[num])))
        num += 1
    return "".join(secret)
```

1.5 Test :

```python
tips= "1234567"
key= "well"
secret = xor_encrypt(tips,key)
print( "cipher_text:", secret )
plaintxt = xor_decrypt( secret, key )
print( "plain_text:",plaintxt )
```

Output:

```
cipher_text: FW_XBS[
plain_text: 1234567
```

**2. P as string, to ADD every charactor to encode and SUB every charactor to decode for every substring of S at the length of P.**

2.1 In fact, the differences between question 1 and question 2 are very small. And I just change 2 XOR character into 1 ADD and 1 SUB in the tow functions.

2.2 Code in Python :

```python
def add_encrypt(tips, key):
    lkey = len(key)
    secret = []
    num = 0
    for each in tips:
        if num >= lkey:
            num = num % lkey
        secret.append(chr(ord(each) + ord(key[num])))
        num += 1

    #return b64.b64encode( "".join( secret ).encode() ).decode()
    return "".join(secret)
```

```python
def sub_decrypt(secret,key):
    #tips = b64.b64decode( secret.encode() ).decode()
    tips = secret
    lkey = len(key)
    secret = []
    num = 0
    for each in tips:
        if num >= lkey:
            num = num % lkey
        secret.append(chr(ord(each) - ord(key[num])))
        num += 1
    return "".join(secret)
```

2.3 Test :

```python
tips = "1234567"
key = "well"
secret = add_encrypt(tips, key)
print("cipher_text:", secret)
plaintxt = sub_decrypt(secret, key)
print("plain_text:", plaintxt)
```

Output :

```
cipher_text: ¨     ¬  £
plain_text: 1234567
```