

## WEEK– 01

### Task – 1

#### Aim:

To implement an XOR-based symmetric encryption and decryption algorithm in Python using a fixed key to transform each character of the text.

#### Description:

The XOR cipher is a simple encryption technique where each character of the plaintext is XORed with a fixed key. The same operation is applied to decrypt the ciphertext because XOR is a reversible operation.

- Key point: Encryption and decryption use the same process.
- Property:  $(\text{Plaintext} \oplus \text{Key}) \oplus \text{Key} = \text{Plaintext}$
- Non-alphabetic characters are also encrypted, as the operation is applied at the byte level.

#### Algorithm:

Input: A plaintext string and a fixed XOR key.

Steps:

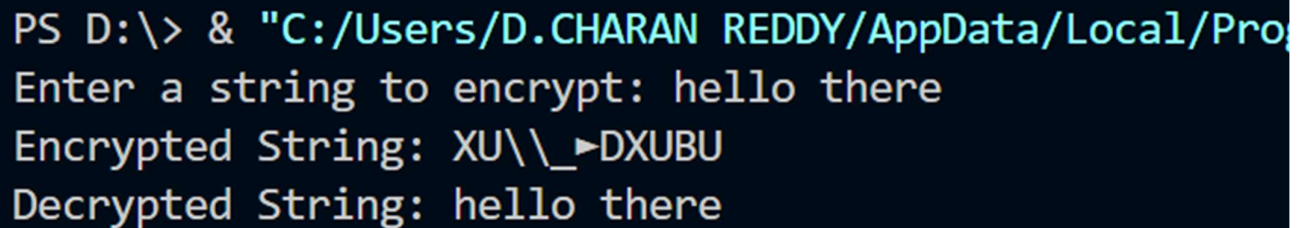
1. Set a fixed key (e.g., 0x0F).
2. For encryption:
  - For each character in the plaintext, convert it to ASCII.
  - XOR it with the key to produce the encrypted value.
  - Convert back to character and append to ciphertext.
3. For decryption:
  - For each character in the ciphertext, repeat the same XOR operation with the same key.
  - Convert back to character to retrieve the original message.

#### Code:

```
s=input("Enter a string to encrypt: ")
encrypted=""
for ch in s:
    new=chr(ord(ch)^ord('0'))
    encrypted+=new
```

```
print('Encrypted String:', encrypted)
decrypted=""
for ch in encrypted:
    new=chr(ord(ch)^ord('0'))
    decrypted+=new
print('Decrypted String:', decrypted)
```

**Output:**



```
PS D:\> & "C:/Users/D.CHARAN REDDY/AppData/Local/Programs/Python/Python310/python.exe"
Enter a string to encrypt: hello there
Encrypted String: XU\_\_DXUBU
Decrypted String: hello there
```

**Result Analysis:**

- The plaintext "hello there" is XORed with the key 0x0F, producing an unreadable encrypted string.
- Decrypting the ciphertext using the same key returns the original plaintext exactly.
- This confirms the correctness of the XOR symmetric cipher as a reversible transformation.

**Conclusion:**

The XOR cipher is simple and efficient, demonstrating the principle of symmetric encryption. Although not secure for real-world use (due to fixed and small key size), it is useful for understanding reversible encryption techniques.

స్వయం తేజస్విన్ భవ

1979

## TASK-02

### Aim:

To implement an encryption algorithm in Python that uses bitwise AND, XOR, and modular arithmetic to transform each character of a string into an encoded output.

### Description:

This algorithm applies a combination of **bitwise operations** and **modular arithmetic** to encrypt characters:

- **Bitwise AND (with 127)** ensures ASCII values are in a valid range.
- **Bitwise XOR (with 127)** flips bits for obfuscation.
- **Modulo 95 + 32** ensures output characters remain printable (from ASCII 32 to 126).  
Decryption is not explicitly shown but would reverse the transformations.

### Algorithm:

**Input:** A plaintext string.

#### Steps:

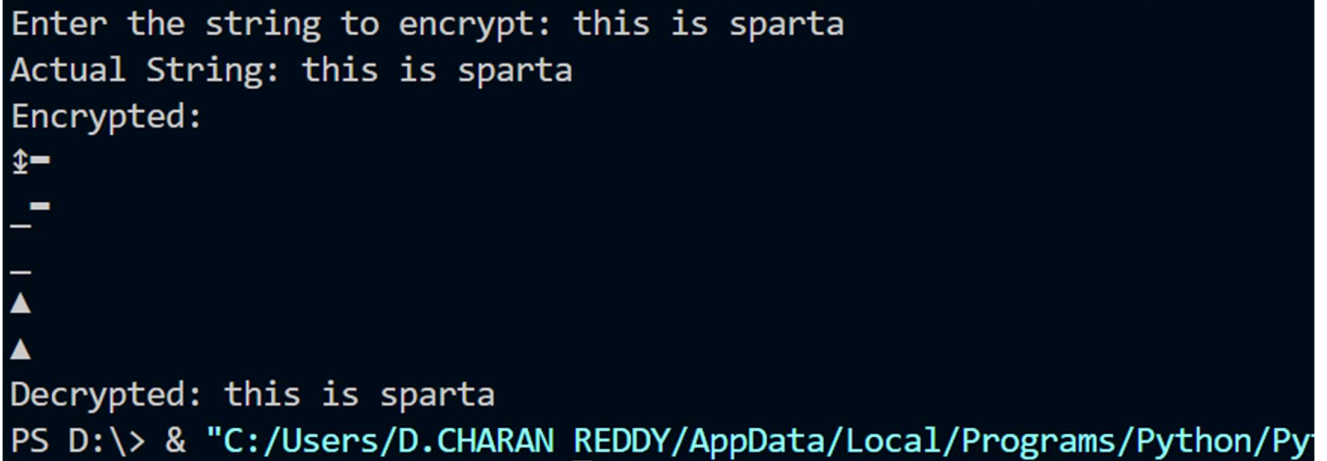
1. Take input string.
2. For each character:
  - Apply  $\text{ord}(\text{char}) \& 127$  to limit ASCII to 7-bit range.
  - XOR the result with 127 to flip bits.
  - Apply modulo operation  $(\text{value} \% 95) + 32$  to ensure printable ASCII.
3. Append transformed character to output string.

### Code:

```
def f(s):  
    res=""  
    for ch in s:  
        a=chr(ord(ch)&127)  
        b=chr(ord(a)^127)  
        res+=b  
    return res  
  
s = input("Enter the string to encrypt: ")  
print("Actual String:", s)  
encrypted=f(s)
```

```
print("Encrypted:",encrypted)
decrypted=f(encrypted)
print("Decrypted:",decrypted)
```

**Output:**



```
Enter the string to encrypt: this is sparta
Actual String: this is sparta
Encrypted:
↑=
-
-
↑
↑
Decrypted: this is sparta
PS D:\> & "C:/Users/D.CHARAN REDDY/AppData/Local/Programs/Python/Py
```

**Result Analysis:**

- The algorithm successfully transforms each character of "this is sparta" into a different printable ASCII character.
- The use of AND, XOR, and modulo operations adds complexity, making the transformation less predictable compared to simple XOR with a fixed key.
- Output remains within readable character set due to  $(\text{value} \% 95) + 32$ .

**Conclusion:**

This experiment demonstrates how bitwise operations can be combined with modular arithmetic for encryption. While still insecure for practical cryptography, it illustrates how mathematical transformations affect character encoding, providing a learning example for advanced obfuscation techniques.