Ex. No. 1	Implement the logical XOR operation to prove its usage in cryptography
Date of Exercise	03.08.2023

Aim

To implement the logical XOR operation to prove its usage in cryptography.

1) Question:

Write a program to check whether a given integer is odd or even.

Description:

This Python program takes an integer as input from the user and determines whether it's an even or odd number. It uses a conditional statement to make this determination and prints the result.

Algorithm:

Step 1: Prompt the user to enter an integer and store it in the variable n.

Step 2: Check whether n^1 is equal to n+1. This part of the code has a logical issue. The ^ operator in Python is a bitwise XOR operator, not the exponentiation operator, so this condition doesn't properly check for even or odd numbers. To fix this issue, you should use the modulus operator (%) to determine whether a number is even or odd.

Step 3: If n % 2 is equal to 0, print "Even."

Step 4: If n % 2 is not equal to 0, print "Odd."

Program:

```
n = int(input("Enter the nubmer: "))

if n^1 == n+1:

    print("Even")

else:

    print("Odd")
```

Output Screenshot:

```
Enter the nubmer:  1
Odd
```

2) Question:

Perform swapping between two integers and display the swapped values.

Description:

This Python program is designed to swap the values of two variables without using a temporary variable. It takes two integer inputs from the user, initially prints their values before the swap, performs the swap operation, and then prints the values after the swap.

Algorithm:

Step 1: Prompt the user to enter the first integer and store it in the variable num1.

Step 2: Prompt the user to enter the second integer and store it in variable num2.

Step 3: Print "Before Swap!" to indicate that the original values will be displayed.

Step 4: Print the values of num1 and num2 to show their initial values.

Step 5: Perform the swap operation without using a temporary variable:

Step 6: Print "After Swap!" to indicate that the values have been swapped.

Step 7: Print the new values of num1 and num2 to show the swapped values.

Program:

```
num1 = int(input("Enter the first number: "))

num2 = int(input("Enter the second number: "))


print("\nBefore Swap!")

print("Num1: ", num1)

print("Num2: ", num2)


num1 = num1 + num2

num2 = num1 - num2

num1 = num1 - num2


print("\nAfter Swap!")
```

```
print("Num1: ", num1)

print("Num2: ", num2)
```

Output Screenshot:

```
Enter the first number:  10
Enter the second number:  20

Before Swap!
Num1:  10
Num2:  20

After Swap!
Num1:  20
Num2:  10
```

3) Question:

Write a program that contains a string (char pointer) with a value\Hello World'. The program should XOR each character in this string with 0, 127 and display the result.

Description:

This Python program defines a function called xor_with_value that performs an XOR (exclusive OR) operation between each character in a given string and a specified integer value. It then demonstrates the use of this function by applying XOR operations with two different values (0 and 127) to a string.

Algorithm:

Step 1: Define the xor_with_value function, which takes two parameters: char_pointer (a string) and value (an integer). The function applies the XOR operation between each character in char_pointer and the provided value.

Step 2: inside the if __name__ == "__main__" block (which ensures the code is only executed when the script is run and not when it's imported as a module):

Create a string called char_pointer with the value "Hello World."

Step 3: Print the original string, which is "Hello World."

Step 4: Call the xor_with_value function with char_pointer and two different values

Program:

```
def xor_with_value(char_pointer, value):
    result = ''.join(chr(ord(char) ^ value) for char in char_pointer)
    return result


if __name__ == "__main__":
    char_pointer = "Hello World"


    print("Original string:", char_pointer)


    xor_with_0 = xor_with_value(char_pointer, 0)
```

```
print("XOR with 0:", xor_with_0)


xor_with_127 = xor_with_value(char_pointer, 127)

print("XOR with 127:", xor_with_127)
```

Output Screenshot:

```
Original string: Hello World
XOR with 0: Hello World
□□R with 127: 7□□□□_(□
```

4) Question:

Prove the security feature when one-time padding operation is performed using logical XOR

Description:

This Python program demonstrates a basic encryption and decryption technique using the XOR (exclusive OR) operation. It defines two functions: encrypt and decrypt, which are used to encrypt and decrypt a given plaintext using a provided key. The program then applies these functions to a sample plaintext and key, showing how to encrypt and subsequently decrypt the text.

Algorithm:

Step 1: encrypt(plaintext, key):

Step 2: decrypt(encrypted_text, key):

Step 3: Inside the if __name__ == "__main__" block (which ensures the code is only executed when the script is run and not when it's imported as a module):

Define the plaintext variable with the value "Hello World."

Define the key variable with the value "Norandomkey." Note that for security, a truly random key should be used, and it should be the same length as the plaintext.

Step 4: Print the plaintext and key to display their values.

Step 5: Encrypt the plaintext using the encrypt function with the provided key, and store the result in the encrypted_text variable.

Step 6: Print the encrypted text, which is the result of the encryption.

Step 7: Decrypt the encrypted text using the decrypt function with the same key, and store the result in the decrypted_text variable.

Step 8: Print the decrypted text, which should match the original plaintext, demonstrating the reversibility of the XOR operation with the same key.

Program:

```
def encrypt(plaintext, key):
    encrypted_text = ''.join(chr(ord(plain_char) ^ ord(key_char)) for plain_char, key_char in zip(plaintext, key))
    return encrypted_text
```

```
def decrypt(encrypted_text, key):
    decrypted_text = ".join(chr(ord(encrypted_char) ^ ord(key_char)) for encrypted_char,
key_char in zip(encrypted_text, key))
    return decrypted_text


if __name__ == "__main__":
    plaintext = "Hello World"
    key = "Norandomkey"  # The key should be truly random and the same length as the plaintext


    print("Plaintext:", plaintext)
    print("Key:", key)


    # Encrypt the plaintext using XOR with the key
    encrypted_text = encrypt(plaintext, key)
    print("Encrypted Text:", encrypted_text)


    # Decrypt the encrypted text using XOR with the same key
    decrypted_text = decrypt(encrypted_text, key)
    print("Decrypted Text:", decrypted_text)
```

Output Screenshot:

```
Plaintext: Hello World
Key: Norandomkey
Encrypted Text: ▯
▯D8▯▯   ▯
Decrypted Text: Hello World
```

Result

Thus, the experiment to Implement the logical XOR operation to prove its usage in cryptography is carried out successfully and obtained the required output.