

## IS EXPLANATION

Write a Java/C/C++/Python program that contains a string (char pointer) with a value \Hello World9. The program should AND OR and XOR each character in this string with 127 and display the result

```
string = input("Enter String: ")

res_and = ""

res_xor = ""

for char in string:
    and_val = ord(char) & 127
    xor_val = ord(char) ^ 127
    res_and += chr(and_val)
    res_xor += chr(xor_val)

print("Original String:", string)
print("AND Result:", res_and)
print("XOR Result:", res_xor)
```

- `ord(char)` gives ASCII value.
- `chr(...)` converts back to character.
- `& 127` performs bitwise AND.
- `^ 127` performs bitwise XOR

### Line-by-Line Explanation

```
python
```

CopyEdit

```
string = input("Enter String: ")
```

- This line takes **input** from the user and stores it in the variable string.

```
python
```

CopyEdit

```
res_and = ""
```

```
res_xor = ""
```

- These are empty strings where we will build the results of AND and XOR operations.

```
python
```

CopyEdit

```
for char in string:
```

- A loop to go through **each character** (char) in the input string.

```
python
```

CopyEdit

```
and_val = ord(char) & 127
```

- `ord(char)` converts the character to its **ASCII value**.
- `& 127` performs a **bitwise AND** with 127 (binary 01111111) — it clears the 8th bit (if any), keeping the lower 7 bits.

```
python
```

CopyEdit

```
xor_val = ord(char) ^ 127
```

- `^ 127` performs a **bitwise XOR** — it flips the bits where 127 has 1. This inverts the lower 7 bits.

```
python
```

CopyEdit

```
res_and += chr(and_val)
```

```
res_xor += chr(xor_val)
```

- `chr(...)` converts the new ASCII values back to **characters**.
- These characters are added (concatenated) to the result strings.

```
python
```

CopyEdit

```
print("Original String:", string)
print("AND Result:", res_and)
print("XOR Result:", res_xor)
```

- Finally, the program prints:
  - The **original input**
  - The string after each character is **AND-ed** with 127
  - The string after each character is **XOR-ed** with 127

### Example:

If you enter:

nginx

CopyEdit

Hello

ASCII of 'H' is  $72, 72 \& 127 = 72, 72 \wedge 127 = 55$

So it builds:

- AND Result: "Hello" (usually unchanged unless 8th bit is set)
- XOR Result: something like "7\x1a\x17\x17\x18" (non-printable or strange chars)

हो, खाली तुझ्या Python कोडचे एकदम सोप्या मराठीत स्पष्टीकरण दिलं आहे:

---

### कोड:

python

CopyEdit

```
string = input("Enter String: ")
res_and = ""
```

```
res_xor = ""

for char in string:
    and_val = ord(char) & 127
    xor_val = ord(char) ^ 127
    res_and += chr(and_val)
    res_xor += chr(xor_val)
```

```
print("Original String:", string)
print("AND Result:", res_and)
print("XOR Result:", res_xor)
```

---

🔍 लाइन बाय लाइन मराठीत स्पष्टीकरण:

---

✓ **string = input("Enter String: ")**

→ वापरकर्त्याकडून एक स्ट्रिंग (म्हणजेच शब्द) घ्यायचा आहे.  
उदा. "Hello World"

---

✓ **res\_and = ""**

→ इथे आपण एक रिकामी स्ट्रिंग तयार करतो जे **AND** ऑपरेशनचे **result** साठवेल.

✓ **res\_xor = ""**

→ हे रिकाम स्ट्रिंग **XOR** ऑपरेशनचे **result** साठी आहे.

---

✓ **for char in string:**

→ आपल्याला दिलेल्या string मधील प्रत्येक अक्षरावर (**character**) loop चालवायचा आहे.

---

✓ **ord(char)**

→ **ord()** फंक्शन वापरून आपण त्या अक्षराचा **ASCII** नंबर घेतो.  
उदा. 'A' साठी **ord('A')** = 65

---

 **and\_val = ord(char) & 127**

→ अक्षराच्या ASCII नंबरवर **AND** ऑपरेशन करत आहोत 127 सोबत.  
& म्हणजे **bitwise AND**.

---

 **xor\_val = ord(char) ^ 127**

→ त्याच ASCII नंबरवर **XOR** ऑपरेशन करत आहोत 127 सोबत.  
^ म्हणजे **bitwise XOR**.

---

 **chr(and\_val)** आणि **chr(xor\_val)**

→ **chr()** फंक्शन वापरून आपल्याला परत त्या ASCII नंबरचे **character** मिळते.  
उदा. **chr(65) = 'A'**

---

 **res\_and += chr(and\_val)**

→ मिळालेलं अक्षर **AND result** स्ट्रंगमध्ये जोडतो.

 **res\_xor += chr(xor\_val)**

→ मिळालेलं अक्षर **XOR result** स्ट्रंगमध्ये जोडतो.

---

 **print(...)**

→ शेवटी आपण **original string, AND result** आणि **XOR result** प्रिंट करतो.

---

 एक उदाहरण:

**Input:**

mathematica

CopyEdit

Enter String: A

**ASCII of 'A' = 65**

- $65 \& 127 = 65 \rightarrow 'A'$
- $65 ^ 127 = 62 \rightarrow '>'$

## **Output:**

rust

CopyEdit

Original String: A

AND Result: A

XOR Result: >

questions\_answers = [

("या प्रोग्रामचा उद्देश काय आहे?",

"प्रत्येक अक्षरावर bitwise AND आणि XOR ॲपरेशन करणे 127 सोबत आणि result दाखवणे. ही टेक्निक डेटा लपवण्यासाठी उपयोगी आहे."),

("आपण 127 का वापरतो?",

"127 हे बायनरीमध्ये 01111111 असते. AND ॲपरेशन 8वा बिट काढून टाकते. XOR ॲपरेशन पहिले 7 बिट्स flip करते. हे बिट लेबल एनक्रिप्शन दाखवते."),

("AND आणि XOR मध्ये काय फरक आहे?",

"AND फक्त ठारिक बिट ठेवतो. XOR बिट्स flip करतो. XOR हे रिहर्सेबल आहे त्यामुळे एनक्रिप्शन साठी उपयोगी.").

("XOR एनक्रिप्शनमध्ये कसा उपयोगी ठरतो?",

"XOR वापरून original डेटा encrypt आणि decrypt करता येतो. कारण XOR रिहर्सेबल आहे."),

("जर दोन वेळा XOR 127 केलं तर काय होतं?",

"मूळ अक्षर परत मिळतं. कारण  $\text{char} \wedge 127 \wedge 127 = \text{char}$ ."),

("`ord()` आणि `chr()` म्हणजे काय?",

"`ord(char)` = ASCII value मिळवतो. `chr(number)` = त्या नंबरचं character परत मिळवतो.")

]

**EX.NO:2**

**PERFORM ENCRYPTION AND DECRYPTION USING THE METHOD OF  
RAILFENCE-ROW & COLUMN**

**TRANSPOSITION TECHNIQUE.**

**AIM:**

Write a Java/C/C++/Python program to perform encryption and decryption using the method of Transposition technique

```
def main():
    print("\n\t\tRAIL FENCE TECHNIQUE")
    a = input("\n\nEnter the input string: ")

    l = len(a)
    c = [''] * l # For ciphertext
    d = [''] * l # For decrypted text

    # Encryption: Rail Fence Cipher
    j = 0
    for i in range(l):
        if i % 2 == 0:
            c[j] = a[i]
            j += 1
    for i in range(l):
        if i % 2 == 1:
            c[j] = a[i]
            j += 1
    ciphertext = ''.join(c)

    print("\nCiphertext after applying rail fence:")
```

```

print(ciphertext)

# Decryption: Rail Fence Cipher
if l % 2 == 0:
    k = l // 2
else:
    k = (l // 2) + 1

j = 0
for i in range(k):
    d[j] = c[i]
    j += 2
j = 1
for i in range(k, l):  #(k,l) is very imop
    d[j] = c[i]
    j += 2
decrypted_text = ''.join(d)

print("Text after decryption:")
print(decrypted_text)

if __name__ == "__main__":
    main()

```

### Question

1. What is a Transposition Cipher?

### Answer

A transposition cipher is a method of encryption where the **positions of characters are shifted** according to a certain pattern, but the **characters themselves remain unchanged**.

Question	Answer
2. What is the Rail Fence Cipher?	Rail Fence Cipher is a type of transposition cipher where the message is written in a zigzag pattern (rails), and then read row by row to get the ciphertext.
3. How many rails are used in your implementation?	My implementation uses <b>2 rails</b> – one for even-indexed letters and one for odd-indexed letters.
4. Is Rail Fence Cipher a symmetric cipher?	Yes, it is a <b>symmetric cipher</b> , because the <b>same key or technique</b> is used for both encryption and decryption.
5. What is the key in Rail Fence Cipher? The key is the <b>number of rails</b> (rows). In this program, it is fixed as 2.	
6. Is Rail Fence Cipher secure?	No, it is <b>not secure for modern encryption</b> because it's easy to break using pattern analysis or brute force.
7. What type of cryptographic technique is used?	It is a <b>Transposition Cipher</b> – part of <b>classical cryptography</b> .
8. What is the difference between Transposition and Substitution cipher?	In <b>Transposition</b> , characters stay the same but their <b>positions change</b> . In <b>Substitution</b> , characters are replaced by <b>other characters or symbols</b> .
9. Can you increase the number of rails?	Yes, we can generalize the logic to <b>more than 2 rails</b> to increase complexity.
10. What is the time complexity of your code?	The code runs in <b>O(n)</b> time where <b>n = length of the message</b> , because it uses simple loops.
•	

प्रश्न: Transposition Cipher म्हणजे काय?

उत्तर: Transposition Cipher मध्ये अक्षरं बदलत नाहीत, पण त्यांच्या स्थिती (क्रम) मध्ये बदल केला जातो.

प्रश्न: Rail Fence Cipher म्हणजे काय?

उत्तर: हा एक प्रकारचा Transposition Cipher आहे ज्यामध्ये अक्षरं एका "झिगझँग" (रेल्वे रांगा) पद्धतीने मांडली जातात आणि नंतर वरून खाली वाचून Ciphertext तयार केला जातो.

प्रश्न: तुझ्या कोडमध्ये किती रेल्स वापरल्या आहेत?

उत्तर: मी 2 रेल्स वापरल्या आहेत – एक सम क्रमांकासाठी (even index), दुसरी विषम क्रमांकासाठी (odd index).

प्रश्न: Rail Fence Cipher symmetric cipher आहे का?

उत्तर: हो, Rail Fence Cipher हा symmetric cipher आहे कारण Encryption आणि Decryption दोन्हीसाठी एकाच तंत्राचा वापर होतो.

प्रश्न: या Cipher मध्ये 'key' म्हणजे काय?

उत्तर: Key म्हणजे rails ची संख्या, म्हणजेच आपण अक्षरं किती रांगेत लिहितो. इथे key = 2 आहे.

प्रश्न: Rail Fence Cipher secure cipher आहे का?

उत्तर: नाही, ही cipher आधुनिक सुरक्षेसाठी पुरेशी नाही. ती सोप्या pattern analysis ने तोडता येते.

प्रश्न: हा cipher कोणत्या प्रकारात मोडतो?

उत्तर: हा cipher Transposition Cipher प्रकारात मोडतो.

प्रश्न: Transposition आणि Substitution Cipher यामध्ये काय फरक आहे?

उत्तर: Transposition Cipher मध्ये अक्षरांची स्थिती बदलते, पण अक्षरं तीच राहतात. Substitution Cipher मध्ये अक्षरं इतर अक्षरांमध्ये बदलली जातात.

प्रश्न: Rails ची संख्या वाढवू शकतो का?

उत्तर: हो, rails ची संख्या वाढवता येते आणि त्यामुळे cipher अजून complex बनतो.

प्रश्न: हा कोड कोणत्या वेळेच्या गुंतागुंतीत चालतो?

उत्तर: हा कोड O(n) वेळेत चालतो, कारण प्रत्येक अक्षर एकदाच हाताळलं जातं.

```
def main():
```

→ main() ही मुख्य फंक्शन आहे जी सगळं encryption/decryption काम करते.

python

Copy

Edit

```
print("\n\t\tRAIL FENCE TECHNIQUE")
```

```
a = input("\n\nEnter the input string: ")
```

→ वापरकर्त्याकडून एक इनपुट स्ट्रिंग घेतली जाते – ज्याचं encryption करायचं आहे.

python

Copy

Edit

```
l = len(a)
```

```
c = [""] * l # For ciphertext  
d = [""] * l # For decrypted text
```

- 1 ही string ची लांबी ठेवते.
- c ही encryption नंतर तयार होणाऱ्या cipher साठी आहे.
- d ही decrypted string ठेवण्यासाठी आहे.

### Encryption Logic

[python](#)

[Copy](#)

[Edit](#)

```
j = 0  
for i in range(l):  
    if i % 2 == 0:  
        c[j] = a[i]  
        j += 1
```

- प्रथम, even-indexed अक्षरं घेतली जातात (0, 2, 4...) आणि c मध्ये भरली जातात.

[python](#)

[Copy](#)

[Edit](#)

```
for i in range(l):  
    if i % 2 == 1:  
        c[j] = a[i]  
        j += 1
```

- नंतर, odd-indexed अक्षरं घेतली जातात (1, 3, 5...) आणि c मध्ये भरली जातात.

[python](#)

[Copy](#)

[Edit](#)

```
ciphertext = ".join(c)
print("\nCiphertext after applying rail fence:")
print(ciphertext)
```

→ c लिस्टला एकत्र करून एक string बनवली जाते आणि ती encryption नंतरची output असते.

### 💡 Decryption Logic

python

Copy

Edit

```
if l % 2 == 0:
```

```
    k = l // 2
```

```
else:
```

```
    k = (l // 2) + 1
```

→ Ciphertext ला दोन भागात विभागायचं असतं – म्हणून k ही मध्याची position मोजली जाते. → सम लांबी असेल तर दोन सारख्या भागांत विभागतो. → विषम लांबी असेल तर पहिला भाग थोडा मोठा.

python

Copy

Edit

```
j = 0
```

```
for i in range(k):
```

```
    d[j] = c[i]
```

```
    j += 2
```

→ Ciphertext चा पहिला भाग (even-position) d च्या 0, 2, 4,... या positions ला ठेवतो.

python

Copy

Edit

```
j = 1
```

```
for i in range(k, l):
```

```
d[j] = c[i]
```

```
j += 2
```

→ Ciphertext चा दुसरा भाग (odd-position) d च्या 1, 3, 5,... या positions ला ठेवतो.

python

Copy

Edit

```
decrypted_text = ".join(d)  
print("Text after decryption:")  
print(decrypted_text)
```

→ d लिस्टला एकत्र करून final plaintext तयार केला जातो आणि print केला जातो.

python

Copy

Edit

```
if __name__ == "__main__":  
    main()
```

→ ही ओळ main() function call करते जेव्हा स्क्रिप्ट थेट चालवली जाते.

**13**

**EX.NO:5**

**IMPLEMENTATION OF RSA**

**AIM:**

**To write a C program to implement the RSA encryption algorithm.**

```
import math
```

```
# Step 1: Input two prime numbers
```

```
p = int(input("Enter a prime number p: "))
```

```
q = int(input("Enter a prime number q: "))
```

```
# Step 2: Calculate n and Euler's totient function phi
```

```
n = p * q
```

```
phi = (p - 1) * (q - 1)
```

```
# Step 3: Choose public exponent e such that 1 < e < phi and gcd(e, phi) = 1
```

```
e = 2
```

```
while e < phi:
```

```
    if math.gcd(e, phi) == 1:
```

```
        break
```

```
    e += 1
```

```
# Step 4: Calculate private exponent d such that (d * e) % phi = 1
```

```
k = 2 # arbitrary small integer to help find d
```

```
d = ((k * phi) + 1) // e
```

```
# Step 5: Display the public and private keys
```

```
print("\nPublic key: (", e, ",", n, ")")
```

```
print("Private key: (", d, ",", n, ")\n")

# Step 6: Input message to encrypt (must be less than n)
msg = int(input("Enter a number message to encrypt (less than n): "))

# Step 7: Encrypt the message
cipher = pow(msg, e, n)
print("Encrypted message:", cipher)

# Step 8: Decrypt the message
original = pow(cipher, d, n)
print("Decrypted message:", original)
```

## RSA Algorithm Overview

RSA (named after Rivest–Shamir–Adleman) is a public-key cryptographic algorithm used for secure data transmission. It uses two keys:

- A public key to encrypt data (shared with everyone).
  - A private key to decrypt data (kept secret).
- 

### Steps in the RSA Algorithm

#### 1. Key Generation

This is done only once and is the foundation of RSA.

1. Choose two large prime numbers  $p$  and  $q$ .
2. Compute:
  - $n = p * q \rightarrow$  This is part of both the public and private key.
  - $\varphi(n) = (p - 1) * (q - 1) \rightarrow$  Euler's totient function.
3. Choose a number  $e$  such that:
  - $1 < e < \varphi(n)$
  - $\gcd(e, \varphi(n)) = 1 \rightarrow e$  is coprime with  $\varphi(n)$
4. Compute the modular inverse of  $e$  mod  $\varphi(n)$ , i.e.,
  - Find  $d$  such that  $(d * e) \% \varphi(n) = 1$
  - This  $d$  is the private key exponent

### Now you have:

- Public Key:  $(e, n) \rightarrow$  Used to encrypt
  - Private Key:  $(d, n) \rightarrow$  Used to decrypt
- 

### Encryption

To encrypt a message  $M$  (as a number, where  $M < n$ ):

$$C = M \text{mod } n \\ C = M^e \text{mod } n$$

Where:

- $C$  is the ciphertext
  - $e$  and  $n$  are from the public key
-

## Decryption

To decrypt ciphertext C:

$$M = C^{d^{-1}} \pmod{n}$$

Where:

- d and n are from the private key
  - The result is the original message M
- 

## Example

Let's say:

- $p = 7, q = 11 \rightarrow \text{So } n = 77, \phi(n) = 60$
- Choose e = 17 (since  $\gcd(17, 60) = 1$ )
- Find d = 53 (since  $(53 * 17) \% 60 = 1$ )

Now:

- Public Key = (17, 77)
- Private Key = (53, 77)

Encrypt message M = 20:

- $C = 20^{17} \% 77 = 50$

Decrypt:

- $M = 50^{53} \% 77 = 20$

 Original message recovered!

## RSA Code Explanation

This program implements a basic RSA algorithm in Python using small prime numbers.

---

### Step 1: Input two prime numbers

**python**

**CopyEdit**

```
p = int(input("Enter a prime number p: "))
```

```
q = int(input("Enter a prime number q: "))
```

- You input two prime numbers, p and q.
  - These are used to generate the keys for RSA.
- 

### Step 2: Calculate modulus n and Euler's totient function phi

**python**

**CopyEdit**

```
n = p * q
```

```
phi = (p - 1) * (q - 1)
```

- $n = p * q$ : This is part of both the public and private key.
  - $\phi = (p - 1) * (q - 1)$ : This is Euler's Totient Function, needed for calculating keys.
- 

### Step 3: Choose the public key exponent e

**python**

**CopyEdit**

```
e = 2
```

```
while e < phi:
```

```
    if math.gcd(e, phi) == 1:
```

```
        break
```

```
    e += 1
```

- Start from  $e = 2$  and find the smallest e such that  $\gcd(e, \phi) = 1$ .
- This means e is coprime with phi.

- e becomes the public exponent (used for encryption).
- 

 Step 4: Find the private key exponent d

python

CopyEdit

```
k = 2 # arbitrary integer
```

```
d = ((k * phi) + 1) // e
```

- This is an approximation to find d such that:

$$(d \cdot e) \bmod \phi = 1 \quad (d \cdot e) \bmod \phi = 1$$

- Technically, d should be the modular inverse of e mod phi, but this shortcut works in simple cases when  $((k \times \phi) + 1)$  is divisible by e.

 For reliable use, use the Extended Euclidean Algorithm to calculate d.

---

 Step 5: Display the keys

python

CopyEdit

```
print("\nPublic key: (", e, ", ", n, ")")
```

```
print("Private key: (", d, ", ", n, ")\n")
```

- Public Key: Used to encrypt the message → (e, n)
  - Private Key: Used to decrypt the message → (d, n)
- 

 Step 6: Input a message to encrypt

python

CopyEdit

```
msg = int(input("Enter a number message to encrypt (less than n): "))
```

- You input a number less than n.
  - RSA works on numbers, so normally text is converted to numbers (you're skipping that here for simplicity).
- 

 Step 7: Encrypt the message

python

**CopyEdit**

**cipher = pow(msg, e, n)**

- **RSA encryption formula:**

**C=Memod nC = M<sup>e</sup> \mod nC=Memodn**

- **pow() is Python's efficient way to compute exponentiation modulo.**
  - **Output: Encrypted cipher.**
- 

 **Step 8: Decrypt the message**

**python**

**CopyEdit**

**original = pow(cipher, d, n)**

- **RSA decryption formula:**

**M=Cdmod nM = C<sup>d</sup> \mod nM=Cdmodn**

- **This recovers the original message.**

<b>Concept</b>	<b>Description</b>
$p, q$	<b>Two prime numbers</b>
$n = p * q$	<b>Modulus used in both public and private keys</b>
$\phi$	<b>Totient function used for key calculations</b>
$e$	<b>Public exponent for encryption</b>
$d$	<b>Private exponent for decryption</b>
$C = M^e \% n$	<b>Encryption formula</b>
$M = C^d \% n$	<b>Decryption formula</b>

**Steps:**

 **Key Generation:**

1. Select two distinct prime numbers  $p$  and  $q$ .
2. Compute  $n = p \times q$  — used in both keys.
3. Compute Euler's totient function  $\phi(n) = (p - 1) \times (q - 1)$ .
4. Choose public exponent  $e$  such that  $1 < e < \phi(n)$  and  $\gcd(e, \phi(n)) = 1$ .
5. Compute private exponent  $d$  such that  $(d \times e) \% \phi(n) = 1$ .

 **Encryption:**

Encrypt the plaintext message  $M$  using the formula:

$$C = M^{e \text{ mod } \phi(n)} \mod n$$

 **Decryption:**

Decrypt the ciphertext  $C$  using the formula:

$$M = C^{d \text{ mod } \phi(n)} \mod n$$

**Common Viva Questions:**

1. Why is RSA called asymmetric?

Because it uses two different keys: a public key for encryption and a private key for decryption.

2. What is the role of Euler's totient function in RSA?

It is used to calculate the private key  $d$ , ensuring that encryption and decryption are inverse operations.

3. Why must  $e$  be coprime with  $\phi(n)$ ?

So that a modular inverse exists, allowing us to compute  $d$ .

4. Why should the message be less than  $n$ ?

Because RSA works in modulo  $n$ , the message must be smaller than  $n$  to prevent encryption errors.

5. Can RSA encrypt text?

Yes, but text must first be converted into numeric form (e.g., using ASCII).

## 6. ? Viva मधील संभाव्य प्रश्न (Expected Viva Questions):

प्रश्न

उत्तर

कारण यामध्ये encryption व decryption साठी केगवेगळ्या keys वापरल्या RSA asymmetric का आहे?

जातात.

phi (n) म्हणजे काय?

हे Euler's totient function आहे, जे  $(p-1)(q-1)$  इतके असते.

$e$  आणि phi (n) मध्ये gcd 1 का

कारण यामुळे d शोधण्यासाठी modular inverse मिळू शकतो.

असावा?

n पेक्षा लहान संदेश का पाहिजे?

कारण RSA मध्ये mod n वापरले जाते.

अक्षरी मजकूर encrypt करू शकतो का?

हो, पण आधी तो ASCII किंवा दुसऱ्या पद्धतीने संख्येत convert करावा लागतो.

```

import math

# ◆ math हे Python चे module आहे. यात gcd() (महत्तम सामान्य विभाजक) वापरण्यासाठी आपण हे import करतो.

p = int(input("Enter a prime number p: "))
q = int(input("Enter a prime number q: "))

# ◆ वापरकर्त्यांकडून दोन prime numbers p आणि q घ्यावेत, हे RSA साठी आवश्यक आहे.

n = p * q
phi = (p - 1) * (q - 1)

# ◆ n ही modulus value आहे (public व private key मध्ये वापरली जाते).
# ◆ phi ही Euler's totient function आहे – key निर्माण करताना खूप महत्वाची.

e = 2
while e < phi:
    if math.gcd(e, phi) == 1:
        break
    e += 1

# ◆ e ही public key आहे.
# ◆ gcd(e, phi) = 1 म्हणजे e आणि phi एकमेकांचे सहसम (coprime) आहेत.
# ◆ लहानातला असा e शोधतो जो या अटींना पूर्ण करतो.

k = 2
d = ((k * phi) + 1) // e

# ◆ d ही private key आहे.
# ◆ ही d अशी असावी की  $(d * e) \% \phi = 1$ 
# ◆ k ही एक सोपी trick आहे d मिळवण्यासाठी.

print("\nPublic key: (", e, ", ", n, ")")

```

```

print("Private key: (", d, ",", n, ")")

# ⚫ public key (e, n)

# ⚫ private key (d, n)

# ⚫ दोन्ही key वापरून encryption व decryption करता येते.

msg = int(input("Enter a number message to encrypt (less than n): "))

# ⚫ वापरकर्ता एक message input देतो (तो n पेक्षा लहान असावा).

C = pow(msg, e, n)

print("Encrypted message:", C)

# ⚫ Encryption formula:  $C = (M^e) \text{ mod } n$ 

# ⚫ pow() ही Python ची function आहे जी efficient modular exponentiation करते.

M = pow(C, d, n)

print("Decrypted message:", M)

# ⚫ Decryption formula:  $M = (C^d) \text{ mod } n$ 

# ⚫ encrypted message C वापरून original message M पुन्हा प्राप्त करतो.

# █ निष्कर्ष:

# हा RSA encryption-decryption चा एक सरल कोड आहे.

# तुमचं original message encryption केल्यानंतर cipher text मध्ये रूपांतरित होतं.

# आणि नंतर तेच cipher text वापरून original message परत मिळवता येतं.

```

**EX.NO:6****IMPLEMENTATION OF DIFFIEHELLMAN KEY EXCHANGE ALGORITHM****AIM:**

**TO IMPLEMENT THE DIFFIE-HELLMAN KEY EXCHANGE ALGORITHM  
USING C LANGUAGE.**

```
<!DOCTYPE html>
<html>
<head>
    <title>Diffie-Hellman Key Exchange</title>
</head>
<body>
    <h2>Diffie-Hellman Key Exchange</h2>
    <hr>

    <script>
        // Function to calculate (a^b) % p
        function power(a, b, p) {
            return Math.pow(a, b) % p;
        }

        var P = 11; // Prime number
        var G = 7; // Primitive root

        var a = 4; // Alice's private key
        var b = 3; // Bob's private key
```

```
var x = power(G, a, P); // Alice's public key
var y = power(G, b, P); // Bob's public key

var ka = power(y, a, P); // Alice's secret key
var kb = power(x, b, P); // Bob's secret key

document.write("P (prime): " + P + "<br>");
document.write("G (base): " + G + "<br>");
document.write("Alice's private key: " + a + "<br>");
document.write("Bob's private key: " + b + "<br>");
document.write("Alice's public key: " + x + "<br>");
document.write("Bob's public key: " + y + "<br>");
document.write("Alice's secret key: " + ka + "<br>");
document.write("Bob's secret key: " + kb + "<br>");
document.write("Shared secret key: " + ka + "<br>");

</script>
</body>
</html>
```

**Diffie-Hellman Key Exchange is a method where Alice and Bob create a shared secret key over a public network without directly sharing the key. They each choose a private number, exchange public values, and use them to independently compute the same secret key known only to both of them.**

डिफी-हेलमन की एक्सचेंज ही एक पद्धत आहे ज्यात Alice आणि Bob सार्वजनिक नेटवर्कवरून एकमेकांशी गुप्त की शेअर करतात, पण ती की थेट शेअर करत नाहीत. ते दोघं स्वतःची खासगी संख्या ठेवतात, एकमेकांना सार्वजनिक मूल्य पाठवतात, आणि त्यावरून एकच सामायिक गुप्त की तयार करतात – जी फक्त त्यांनाच माहिती असते.

## Aim:

To implement the Diffie-Hellman Key Exchange algorithm using C language.

```
<html>
<head>
    h2>Diffie-Hellman Key Exchange
</head>
<hr>
<script>          // Function to calculate (a-b) % p
    functo power(a, b, p) {
        return Math.pow(a, b) % p;      Prime number
    }
    var P = 11                  // Prime number
    var G = 7                  // Primitive rrect
    var x = a= power(G, a, P);  Alice's privac key
    var y = power(G, b, P);    Bob's public key
    var ka= power(y, a, P);   Alice's secret key
    var kb= power(x, b, P);   Bob's secret key
    document.write("P (prime): + " + h "<br> ");
    document.write("G (base): " + Ar base); + " + br"\br\");
    document.write("Alice's private key: " + " + br"\br\");
    document.write("Bob's public key: " + " + br"\br\");
    document.write("Alice's secret key: " + " + br"\br\");
    document.write("Snlarha secret key: " + " + br"\br\");
</script>
</body>
</html>
```

## Diffie-Hellman Key Exchange Viva Questions (English)

**Q1:** What is the use of the Diffie-Hellman algorithm?

**A:** It is used to securely exchange a secret key between two parties.

**Q2:** Does Diffie-Hellman perform encryption?

**A:** No, it is used only for key exchange, not encryption.

**Q3:** What are the public and private components in Diffie-Hellman?

**A:** Public: P (prime) and G (base); Private: Alice's a, Bob's b.

**Q4:** What is a primitive root?

**A:** A number G is a primitive root modulo P if its powers ( $G^k \bmod P$ ) generate all the integers from 1 to P-1.

**Q5:** How is the secret key generated?

**A:** Alice generates:  $K = y^a \bmod P$ , Bob generates:  $K = x^b \bmod P$ ; both parties end up with the same key.

**Q6:** Why is Diffie-Hellman considered secure?

**A:** It uses the **Discrete Logarithm Problem**, which is difficult to solve efficiently.

**Q7:** Can a hacker obtain the shared secret key if they only know the public keys?

**A:** No, because they don't have the private keys.

**Q8:** What is the weakness of the Diffie-Hellman algorithm?

**A:** **Man-in-the-Middle Attack** can occur if proper authentication is not used.

**Q9:** Why do both parties end up with the same secret key?

**A:** Both parties end up with the same key because  $G^{(ab)} \bmod P$  results in the same value for both.

**Q10:** In which encryption methods is the secret key used?

**A:** The secret key is used in **symmetric encryption** methods like AES or DES.

## **Diffie-Hellman Key Exchange Viva Questions (मराठी)**

**प्र.1:** Diffie-Hellman अल्गोरिदमचा उपयोग काय आहे?

**उत्तर:** दोन पक्षांमध्ये गुप्त की सुरक्षितपणे शेअर करण्यासाठी वापरला जातो.

**प्र.2:** Diffie-Hellman मध्ये एन्क्रिप्शन होते का?

**उत्तर:** नाही, हा अल्गोरिदम फक्त गुप्त की शेअर करण्यासाठी वापरला जातो.

**प्र.3:** सार्वजनिक आणि खाजगी घटक कोणते असतात?

**उत्तर:** सार्वजनिक: P (prime) आणि G (base); खाजगी: Alice ची a, Bob ची b.

**प्र.4:** Primitive root म्हणजे काय?

**उत्तर:** P च्या प्रत्येक अव्यक्त संख्येसाठी  $(\text{mod } P)$  जी संख्येची ताकद घेऊन 1 ते  $P-1$  पर्यंत सर्व संख्यांचा समावेश होतो, ती G primitive root असते.

**प्र.5:** Secret key कशी तयार होते?

**उत्तर:** Alice कडून:  $K = y^a \text{ mod } P$ , Bob कडून:  $K = x^b \text{ mod } P$ ; दोघांना एकसारखी key मिळते.

**प्र.6:** Diffie-Hellman सुरक्षित का आहे?

**उत्तर:** कारण यामध्ये Discrete Logarithm Problem वापरली जाते जी सोडवणे खूप कठीण आहे.

**प्र.7:** हेकरने public key पाहिली, तरी तो shared key मिळवू शकतो का?

**उत्तर:** नाही, कारण त्याला private key माहिती नसते.

**प्र.8:** या अल्गोरिदमची कमकुवत बाजू कोणती आहे?

**उत्तर:** Man-in-the-Middle Attack जर authentication नसेल तर शक्य आहे.

**प्र.9:** दोघांची secret key एकसारखी का असते?

**उत्तर:** कारण  $G^{(ab)} \text{ mod } P$  हे दोघांनाही समान key देते.

**प्र.10:** ही secret key कोणत्या encryption मध्ये वापरली जाते?

**उत्तर:** ही symmetric encryption मध्ये वापरली जाते जसे की AES किंवा DES

## What is a Primitive Root?

A **primitive root** of a prime number  $p$  is a number  $g$  such that when raised to different powers, it generates all the numbers from 1 to  $p-1$  (under modulo  $p$ ). In simpler terms, a primitive root is a number that can produce every integer from 1 to  $p-1$  when repeatedly raised to different powers and reduced modulo  $p$ .

### Example:

Consider the prime number  $p = 7$ . We want to find a primitive root for  $p$ .

Let's check for  $g = 3$ :

- $3 \mod 7 = 3$   
 $3^1 \mod 7 = 3$
- $3^2 \mod 7 = 9 \mod 7 = 2$   
 $3^3 \mod 7 = 27 \mod 7 = 6$
- $3^4 \mod 7 = 81 \mod 7 = 4$   
 $3^5 \mod 7 = 243 \mod 7 = 5$

- $36 \bmod 7 = 729 \bmod 7 = 13^6 \bmod 7 = 729 \bmod 7 = 136 \bmod 7 = 729 \bmod 7 = 1$

Here, we see that the powers of **3** produce all the integers from 1 to  $p-1=6$  $p-1=6$  $p-1=6$  $p-1=6$  under modulo 7 (i.e., **3, 2, 6, 4, 5, 1**). So, **3** is a **primitive root** modulo 7.

### Why it's Useful in Cryptography:

In cryptography, primitive roots help generate secure keys. They allow for a wide range of possible key values that make it difficult for attackers to reverse-engineer the keys or find patterns. In systems like **Diffie-Hellman Key Exchange**, primitive roots ensure that both parties can generate the same shared secret key without exposing it directly.

### External Example:

If Alice and Bob use a primitive root (e.g., **g = 3**) and a prime number (e.g., **p = 7**) to exchange keys, they can each calculate a public key based on their private keys. Because of the properties of primitive roots, even though they exchange public keys, the shared secret key they compute will be very difficult for anyone else to guess.

4o mini

### You said:

in marathi

### ChatGPT said:

#### Primitive Root म्हणजे काय?

**Primitive Root** (प्राथमिक मुळ) हे एखाद्या पाइम संख्या **p** साठी एक असे संख्यात्मक गुट (ग) असते, जे वेळोवेळी वेगवेगळ्या घातांकावर (powers) उचलल्यावर 1 ते  $p-1$  $p-1$  $p-1$  $p-1$  (modulo p) दरम्यान सर्व संख्या निर्माण करू शकते. सोप्या भाषेत, primitive root म्हणजे अशी संख्या जी वेगवेगळ्या घातांकावर उचलून **p** ला विभागल्यावर सर्व संभाव्य संख्यांचा सेट निर्माण करते.

### उदाहरण:

धरणे, आपल्याकडे **p = 7** (prime number) आहे, आणि आपल्याला primitive root शोधायचा आहे.

चला, **g = 3** हा नंबर तपासूया:

- $31 \bmod 7 = 33^1 \bmod 7 = 331 \bmod 7 = 3$
- $32 \bmod 7 = 9 \bmod 7 = 23^2 \bmod 7 = 9 \bmod 7 = 232 \bmod 7 = 9 \bmod 7 = 2$
- $33 \bmod 7 = 27 \bmod 7 = 63^3 \bmod 7 = 27 \bmod 7 = 633 \bmod 7 = 27 \bmod 7 = 6$
- $34 \bmod 7 = 81 \bmod 7 = 43^4 \bmod 7 = 81 \bmod 7 = 434 \bmod 7 = 81 \bmod 7 = 4$
- $35 \bmod 7 = 243 \bmod 7 = 53^5 \bmod 7 = 243 \bmod 7 = 535 \bmod 7 = 243 \bmod 7 = 5$
- $36 \bmod 7 = 729 \bmod 7 = 13^6 \bmod 7 = 729 \bmod 7 = 136 \bmod 7 = 729 \bmod 7 = 1$

आता, आपण पाहू शकतो की **3** च्या घातांमुळे **1** ते **6** पर्यंतच्या सर्व संख्यांचा सेट (i.e., **3, 2, 6, 4, 5, 1**) तयार होतो. त्यामुळे, **3** हा **7** साठी एक **primitive root** आहे.

### क्रिप्टोग्राफीमध्ये त्याचे महत्त्व:

क्रिप्टोग्राफीमध्ये primitive roots सुरक्षित की तयार करण्यात मदत करतात. या मुळे, खाजगी आणि सार्वजनिक कोंच्या गणनेची प्रक्रिया गुंतागुंतीची आणि सुरक्षित होते. **Diffie-Hellman Key Exchange** सारख्या सिस्टममध्ये, primitive roots चा वापर सुरक्षित संवाद साधण्यासाठी केला जातो, कारण हे खाजगी कीची मूळे लपवून ठेवून सार्वजनिक कीसाठी एक विस्तृत श्रेणी निर्माण करतात.

#### बाब्य उदाहरण:

समजा, **Alice** आणि **Bob** एक primitive root (उदाहरणार्थ  $g = 3$ ) आणि prime number (उदाहरणार्थ  $p = 7$ ) वापरून की एकसचेंज करीत आहेत. ते दोघेही त्यांचे सार्वजनिक की त्यांच्या खाजगी कीवरून गणना करतात. Primitive roots च्या गुणधर्मामुळे, ते सार्वजनिक की एकसचेंज करतात तरीही त्यांनी एकसारखी गुप्त (shared) की तयार केली आहे, जी इतर कोणालाही शोधता येणे खूप कठीण आहे.

### . JavaScript फंक्शन (Power Function)

javascript

CopyEdit

```
// Function to calculate (a^b) % p
function power(a, b, p) {
    return Math.pow(a, b) % p;
}
```

#### स्पष्टीकरण:

- या फंक्शनमध्ये 3 इनपुट आहेत:  $a$ ,  $b$ , आणि  $p$ .
- $\text{Math.pow}(a, b)$  ही JavaScript ची inbuilt फंक्शन आहे, जी  $a^b$  काढते.
- $\% p$  याचा अर्थ आहे परिणामाला  $p$  ने मोडून काढणे. त्यामुळे हे फंक्शन  $(a^b) \% p$  परत करते.

#### 4. सार्वजनिक मूल्ये सेट करणे (Defining Public Values)

javascript

CopyEdit

```
var P = 11; // Prime number
```

```
var G = 7; // Primitive root
```

स्पष्टीकरण:

- P ही एक **prime number** आहे (11).
  - G ही **primitive root** आहे (7).
    - **Primitive root** ही संख्या जी एक prime number (P) च्या सर्व शक्य मूल्यांना तयार करू शकते.
- 

#### 5. खाजगी की (Private Key) सेट करणे

javascript

CopyEdit

```
var a = 4; // Alice's private key
```

```
var b = 3; // Bob's private key
```

स्पष्टीकरण:

- a आणि b हे खाजगी की आहेत.
    - a म्हणजे Alice ची खाजगी की (4).
    - b म्हणजे Bob ची खाजगी की (3).
- 

#### 6. सार्वजनिक की निर्माण करणे (Calculating Public Keys)

javascript

CopyEdit

```
var x = power(G, a, P); // Alice's public key
```

```
var y = power(G, b, P); // Bob's public key
```

स्पष्टीकरण:

- x आणि y या दोन सार्वजनिक की आहेत, ज्या खालीलप्रमाणे तयार केल्या जातात:
  - Alice च्या सार्वजनिक कीला  $G^a \% P$  ( म्हणजे  $7^{4 \% 11}$  ) म्हणून काढले जाते.
  - Bob च्या सार्वजनिक कीला  $G^b \% P$  ( म्हणजे  $7^{3 \% 11}$  ) म्हणून काढले जाते.

---

## 7. गुप्त की निर्माण करणे (Calculating Shared Secret Key)

javascript

CopyEdit

```
var ka = power(y, a, P); // Alice's secret key
```

```
var kb = power(x, b, P); // Bob's secret key
```

स्पष्टीकरण:

- $ka$  म्हणजे Alice च्या गुप्त कीला  $y^a \% P$  ( म्हणजे  $y = 7^{13} \% 11$  ) म्हणून काढले जाते.
- $kb$  म्हणजे Bob च्या गुप्त कीला  $x^b \% P$  ( म्हणजे  $x = 7^{14} \% 11$  ) म्हणून काढले जाते.
  - इथे, Alice आणि Bob दोघेही एकसारखी गुप्त की मिळवतात, कारण  $ka$  आणि  $kb$  समान असतील.

---

## 8. परिणाम प्रदर्शित करणे (Displaying Results)

javascript

CopyEdit

```
document.write("P (prime): " + P + "<br>");
```

```
document.write("G (base): " + G + "<br>");
```

```
document.write("Alice's private key: " + a + "<br>");
```

```
document.write("Bob's private key: " + b + "<br>");
```

```
document.write("Alice's public key: " + x + "<br>");
```

```
document.write("Bob's public key: " + y + "<br>");
```

```
document.write("Alice's secret key: " + ka + "<br>");
```

```
document.write("Bob's secret key: " + kb + "<br>");
```

```
document.write("Shared secret key: " + ka + "<br>");
```

स्पष्टीकरण:

- `document.write` चा वापर करून सर्व गणना किमान वेब पेजवर प्रदर्शित केल्या जातात.
  - पेजवर  $P$ ,  $G$ , Alice आणि Bob चे खाजगी आणि सार्वजनिक की आणि शेवटी गुप्त (shared) की या सर्व माहिती दिसेल.

---

एकूण प्रक्रिया:

1. **P** आणि **G** (prime आणि primitive root) सार्वजनिकपणे ओळखली जातात.
  2. **a** आणि **b** खाजगी की असतात ज्याचा वापर करून Alice आणि Bob आपली सार्वजनिक की तयार करतात.
  3. त्यानंतर, दोघेही एकमेकांची सार्वजनिक की वापरून गुप्त की तयार करतात.
  4. शेवटी, दोघांनाही एकसारखी गुप्त की मिळते.
- 

#### निष्कर्ष:

Diffie-Hellman Key Exchange चा उपयोग मुख्यतः सुरक्षित संवाद साधण्यासाठी गुप्त की निर्मितीसाठी केला जातो. यामध्ये सार्वजनिक कीच्या आधारे खाजगी की संरक्षित असते.

तुम्हाला अजून काही शंका असतील तर, कृपया विचारू शकता!

### Aim and Objective of the Experiment:

#### Aim:

The aim of this experiment is to understand and implement the **Diffie-Hellman Key Exchange Algorithm**. This algorithm allows two parties to securely exchange a shared secret key over an insecure communication channel.

#### Objectives:

1. **Learn how public and private keys are used:** The experiment demonstrates how two parties, Alice and Bob, can use public and private keys to exchange information securely.
2. **Understand the concept of Primitive Root:** The experiment helps understand how a primitive root (a special number) is used in cryptographic algorithms to generate secure keys.
3. **Generate Shared Secret Key:** It shows how, using public keys and private keys, both Alice and Bob can generate the same shared secret key without directly transmitting it, ensuring privacy.

4. **Learn the process of secure key exchange:** The experiment highlights the role of mathematical operations (modular exponentiation) in ensuring that the key exchange process is secure.

In simple terms, this experiment teaches how two people can securely agree on a secret key, even if they are communicating over a public network, by using math and cryptographic principles.

4o mini

**You said:**

in marathi

**ChatGPT said:**

प्रयोगाचा उद्देश आणि ध्येय:

उद्देश:

हा प्रयोग **Diffie-Hellman Key Exchange Algorithm** समजून घेण्यासाठी आणि त्याचा अंमलबजावणी करण्यासाठी आहे. हा अल्गोरिदम दोन पक्षांना एका असुरक्षित संवाद मार्गावर गुप्त की सुरक्षितपणे बदलण्याची सुविधा प्रदान करतो.

ध्येय:

1. **सार्वजनिक आणि खाजगी कींचा वापर समजून घेणे:** हा प्रयोग दाखवतो की कसे दोन पक्ष, Alice आणि Bob, सार्वजनिक आणि खाजगी की वापरून सुरक्षितपणे माहिती देवाणघेवाण करू शकतात.
2. **Primitive Root संकल्पना समजून घेणे:** हा प्रयोग cryptographic algorithms मध्ये सुरक्षित की तयार करण्यासाठी कशाप्रकारे primitive root (विशेष संख्या) वापरली जाते हे समजून सांगतो.
3. **गुप्त की तयार करणे:** हा प्रयोग दाखवतो की सार्वजनिक आणि खाजगी कींचा वापर करून दोन्ही पक्ष एकसारखी गुप्त की तयार करू शकतात, जी थेट ट्रान्समिट केली जात नाही, म्हणून ती सुरक्षित असते.
4. **सुरक्षित की एक्सचेंज प्रक्रिया समजून घेणे:** हा प्रयोग दर्शवतो की गणिती क्रिया (modular exponentiation) कशाप्रकारे की एक्सचेंज प्रक्रिया सुरक्षित ठेवतात.

सोप्या शब्दात, हा प्रयोग शिकवतो की दोन व्यक्ती एक गुप्त की सुरक्षितपणे कशी तयार करू शकतात, जरी त्यांचा संवाद सार्वजनिक नेटवर्कवर होत असला तरी, गणित आणि क्रिप्टोग्राफिक तत्वांचा वापर करून.