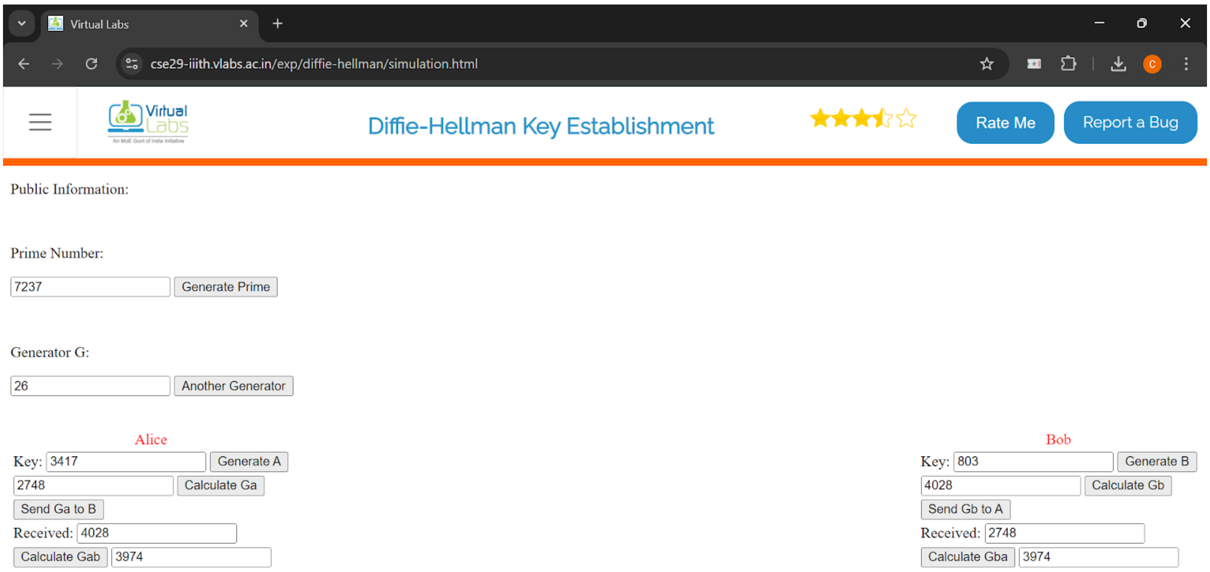
Experiment No. 6

Title: Diffie-Hellman Key Exchange Protocol

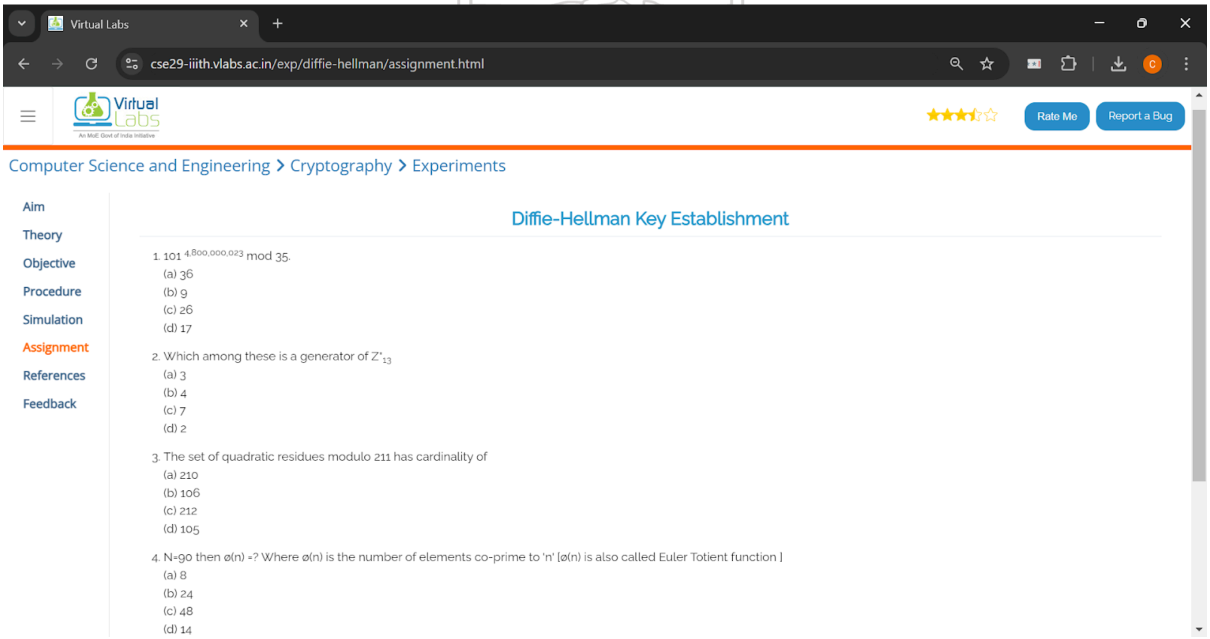
Batch: B-3 **Roll No.: 16010422234** **Experiment No.: 06**

Results:

Simulation:



Assignment:



Q1. (c) 26

We first reduce the exponent modulo the Euler's Totient function $\phi(35)$ because of Euler's theorem, which states that for any a coprime to n,

$$a^{\phi(n)} \equiv 1 \pmod n$$

Where $\phi(35) = \phi(5) \times \phi(7) = 4 \times 6 = 24$.

So, reduce the exponent modulo 24:

$$4,800,000,023 \mod 24 = 23$$

Thus, we need to compute:

$$101^{23} \mod 35$$

Since $101 \equiv 31 \pmod{35}$, we calculate:

$$31^{23} \mod 35$$

This gives us:

$$31^{23} \mod 35 = 26$$

Answer: 26

Q2. (c), (d)

$\mathbb{Z}_{13}^*$ consists of the integers {1, 2, 3, ..., 12} under multiplication modulo 13. A generator g of $\mathbb{Z}_{13}^*$ is an element such that:

$$\{g^1, g^2, \ldots, g^{12}\}$$

yields all elements of $\mathbb{Z}_{13}^*$. By checking possible values, we find that the generators are $2, 6, 7$, and 11.

Answer: Any of $2, 6, 7, 11$

Q3. (d)

For a prime modulus p, the number of quadratic residues is:

$$\frac{p-1}{2}$$

For $p = 211$:

$$\text{Cardinality} = \frac{211-1}{2} = \frac{210}{2} = 105$$

Answer: 105

Q4. (b)

Euler's Totient function $\phi(N)$ is the number of integers up to N that are coprime with N. For $N = 90$:

$$90 = 2 \times 3^2 \times 5$$

So:

$$\phi(90) = 90 \times \left(1 - \frac{1}{2}\right) \times \left(1 - \frac{1}{3}\right) \times \left(1 - \frac{1}{5}\right) = 90 \times \frac{1}{2} \times \frac{2}{3} \times \frac{4}{5} = 24$$

Answer: 24

server.py

```python
from flask import Flask, request, jsonify, send_from_directory
import random
import os

app = Flask(__name__)

# Route to serve the client.html file
@app.route('/')
def serve_client():
    return send_from_directory(os.getcwd(), 'client.html')

# Endpoint for the Diffie-Hellman exchange
```

```
@app.route('/exchange', methods=['POST'])
def exchange_keys():
    data = request.json
    if not data:
        print("No data received")
        return jsonify({"error": "No data received"}), 400

    try:
        p = int(data['p'])
        g = int(data['g'])
        RA = int(data['RA'])

        print(f"Received from client - p: {p}, g: {g}, RA: {RA}")

        b = random.randint(1, p-1)
        print(f"Server's secret key (b): {b}")

        RB = pow(g, b, p)
        print(f"Calculated RB (g^b mod p): {RB}")

        KAB = pow(RA, b, p)
        print(f"Calculated shared key KAB: {KAB}")

        return jsonify({"RB": str(RB)})

    except Exception as e:
        print(f"Error: {str(e)}")
        return jsonify({"error": str(e)}), 500

# Endpoint for secure communication
@app.route('/secure_communication', methods=['POST'])
def secure_communication():
    data = request.json
    try:
        shared_key = int(data['shared_key'])
        print(f"Received shared key from client: {shared_key}")

        encrypted_message = additive_cipher_encrypt("Hello from server!",
shared_key)
        print(f"Encrypted message to send to client: {encrypted_message}")

        # Decrypting the message back to demonstrate decryption
```

```
        decrypted_message = additive_cipher_decrypt(encrypted_message,
shared_key)
        print(f"Decrypted message on server-side: {decrypted_message}")

        return jsonify({"encrypted_message": encrypted_message,
"decrypted_message": decrypted_message})
    except Exception as e:
        print(f"Error: {str(e)}")
        return jsonify({"error": str(e)}), 500

def additive_cipher_encrypt(message, key):
    return ''.join(chr((ord(char) + key) % 256) for char in message)

def additive_cipher_decrypt(encrypted_message, key):
    return ''.join(chr((ord(char) - key) % 256) for char in
encrypted_message)

if __name__ == "__main__":
    print("Server is starting.")
    app.run(debug=True, host='127.0.0.1', port=5500)
    print("Server is running on port 5500.")
```


client.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Diffie-Hellman Simulation</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 20px;
            background-color: #f4f4f4;
        }
        h1 {
            text-align: center;
        }
        .container {
            max-width: 600px;
            margin: 0 auto;
            background: white;
```

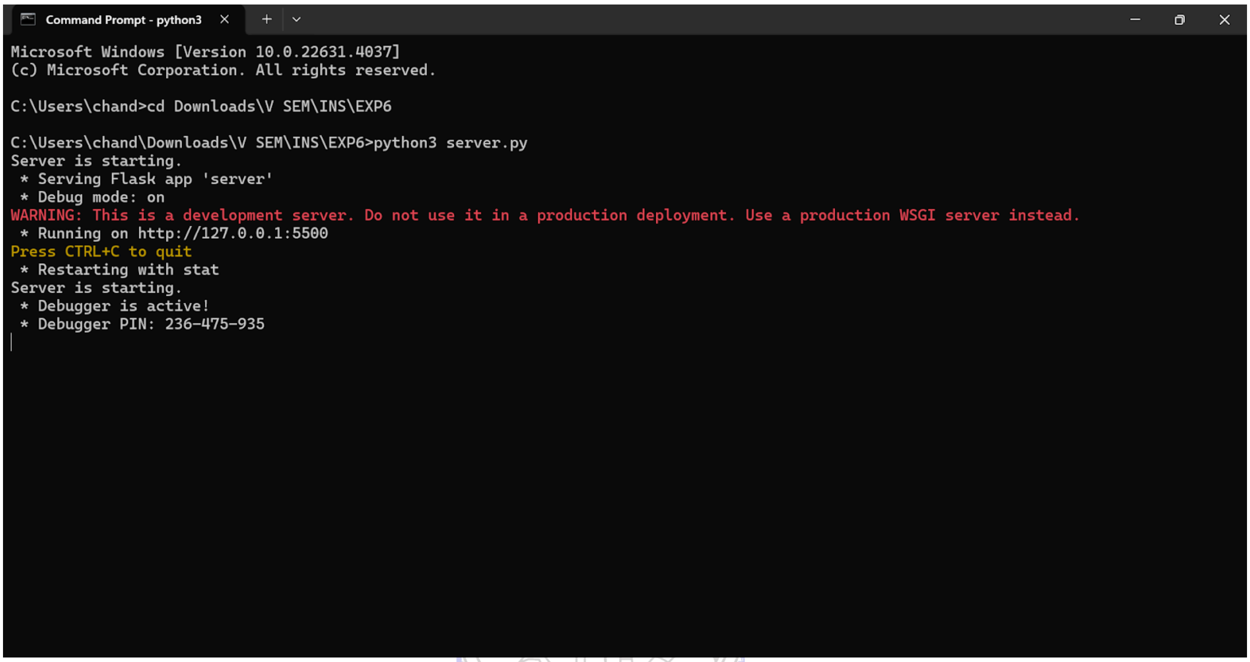
```
            padding: 20px;
            border-radius: 8px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
        }
        .step {
            margin: 20px 0;
        }
        button {
            background-color: #6345a0;
            color: white;
            padding: 10px 15px;
            border: none;
            border-radius: 5px;
            cursor: pointer;
        }
        button:hover {
            background-color: #4c4eaf;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Diffie-Hellman Key Exchange Simulation</h1>
        <div class="step">
            <h2>Step 1: Choose Prime Number (p) and Generator (g)</h2>
            <label for="prime">Prime Number (p):</label>
            <input type="number" id="prime" placeholder="Enter a prime
number" value="23">
            <label for="generator">Generator (g):</label>
            <input type="number" id="generator" placeholder="Enter a
generator" value="5">
            <button onclick="choosePrimeAndGenerator()">Set p and
g</button>
        </div>
        <div class="step">
            <h2>Step 2: Generate Secret Key</h2>
            <button onclick="generateSecret()">Generate Secret for
Client</button>
            <p id="secret"></p>
        </div>
        <div class="step">
            <h2>Step 3: Calculate RA and Send to Server</h2>
            <button onclick="calculateRA()">Calculate RA and Send</button>
```

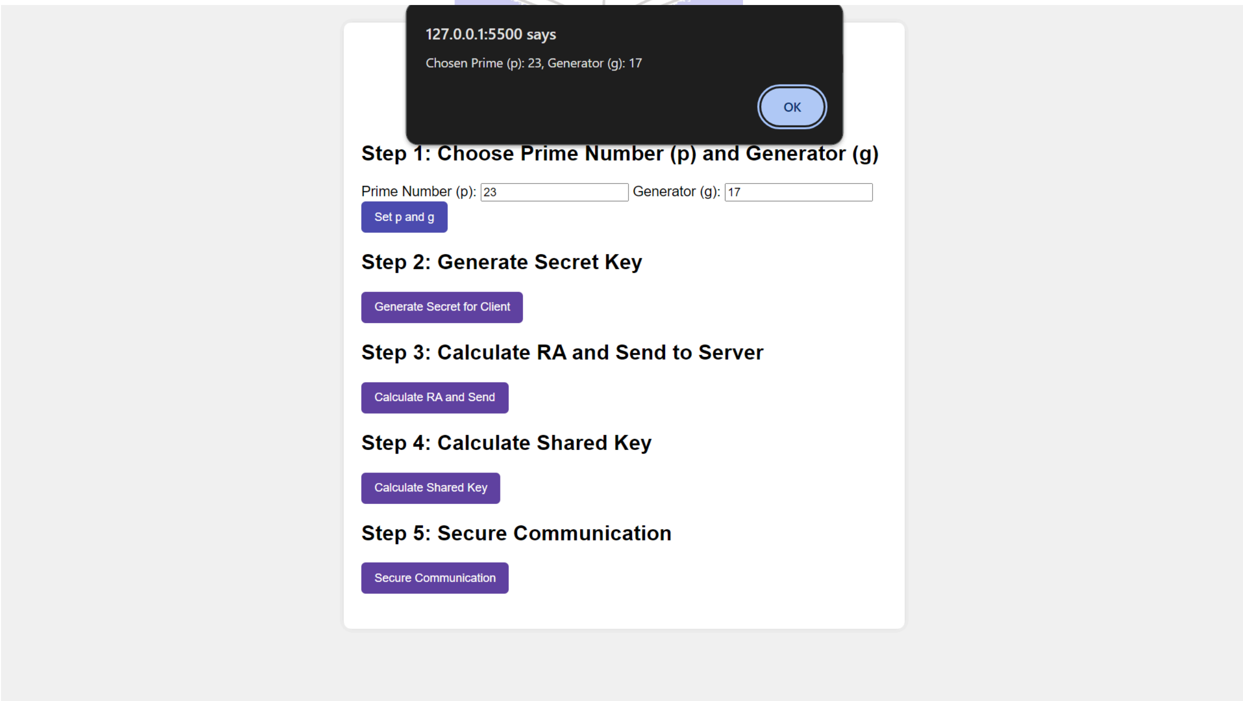
```
            <p id="public-key"></p>
        </div>
        <div class="step">
            <h2>Step 4: Calculate Shared Key</h2>
            <button onclick="calculateSharedKey()">Calculate Shared
Key</button>
            <p id="shared-key"></p>
        </div>
        <div class="step">
            <h2>Step 5: Secure Communication</h2>
            <button onclick="secureCommunication()">Secure
Communication</button>
            <p id="encrypted-message"></p>
            <p id="decrypted-message"></p>
        </div>
    </div>

    <script>
        let p, g, a, RA, RB;

        function choosePrimeAndGenerator() {
            p = BigInt(document.getElementById('prime').value);
            g = BigInt(document.getElementById('generator').value);
            alert(`Chosen Prime (p): ${p}, Generator (g): ${g}`);
        }

        function generateSecret() {
            a = BigInt(Math.floor(Math.random() * (Number(p) - 2)) + 1);
            document.getElementById('secret').innerText = `Client's Secret
(a): ${a}`;
        }

        function calculateRA() {
            RA = g ** a % p;
            document.getElementById('public-key').innerText = `Calculated
RA (g^a mod p): ${RA}`;

            fetch('http://127.0.0.1:5500/exchange', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
```

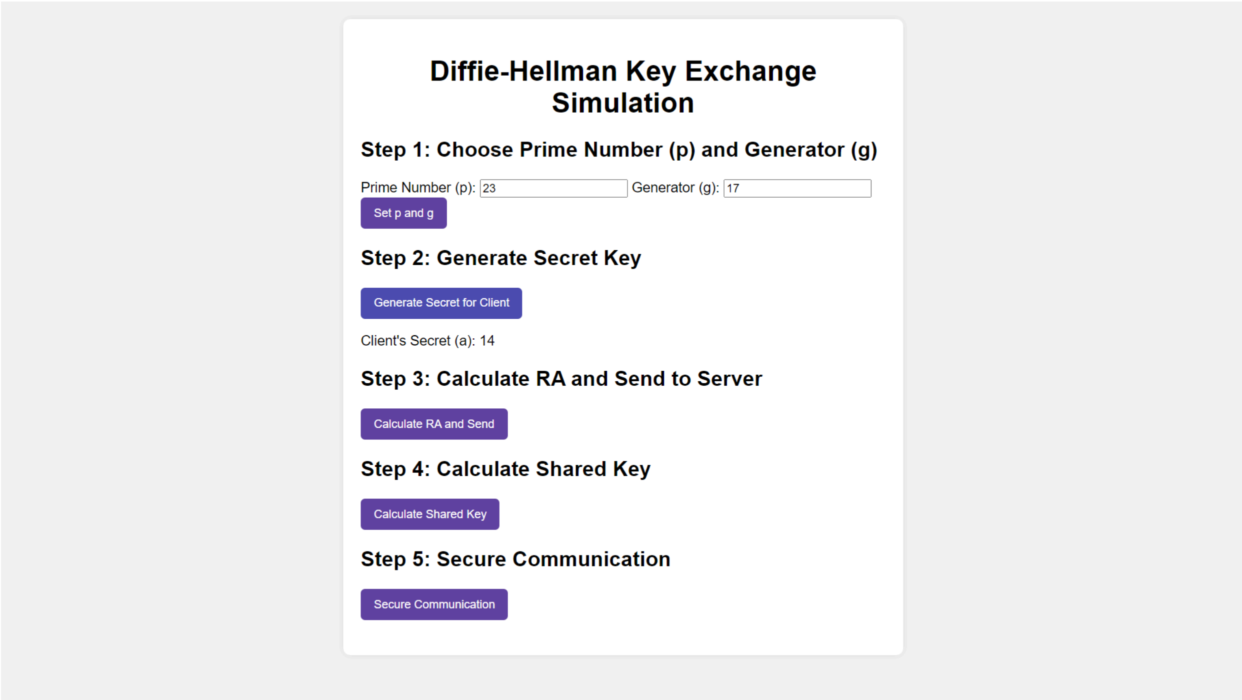
```
                body: JSON.stringify({ p: p.toString(), g: g.toString(),
RA: RA.toString() })
            })
            .then(response => response.json())
            .then(data => {
                if (data.RB) {
                    RB = BigInt(data.RB);
                    alert(`Received RB from server: ${RB}`);
                } else {
                    alert('RB has not been received from the server.');
                }
            })
            .catch(error => {
                console.error('Error:', error);
            });
        }

        function calculateSharedKey() {
            if (RB === undefined) {
                alert('RB has not been received from the server.');
                return;
            }
            let KAB = RB ** a % p;
            document.getElementById('shared-key').innerText = `Shared Key
KAB (Client's perspective): ${KAB}`;
        }

        function secureCommunication() {
            let KAB = RB ** a % p;
            fetch('http://127.0.0.1:5500/secure_communication', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify({ shared_key: KAB.toString() })
            })
            .then(response => response.json())
            .then(data => {
                let encryptedMessage = data.encrypted_message;
                let decryptedMessage = data.decrypted_message;
                document.getElementById('encrypted-message').innerText =
`Encrypted Message: ${encryptedMessage}`;
```

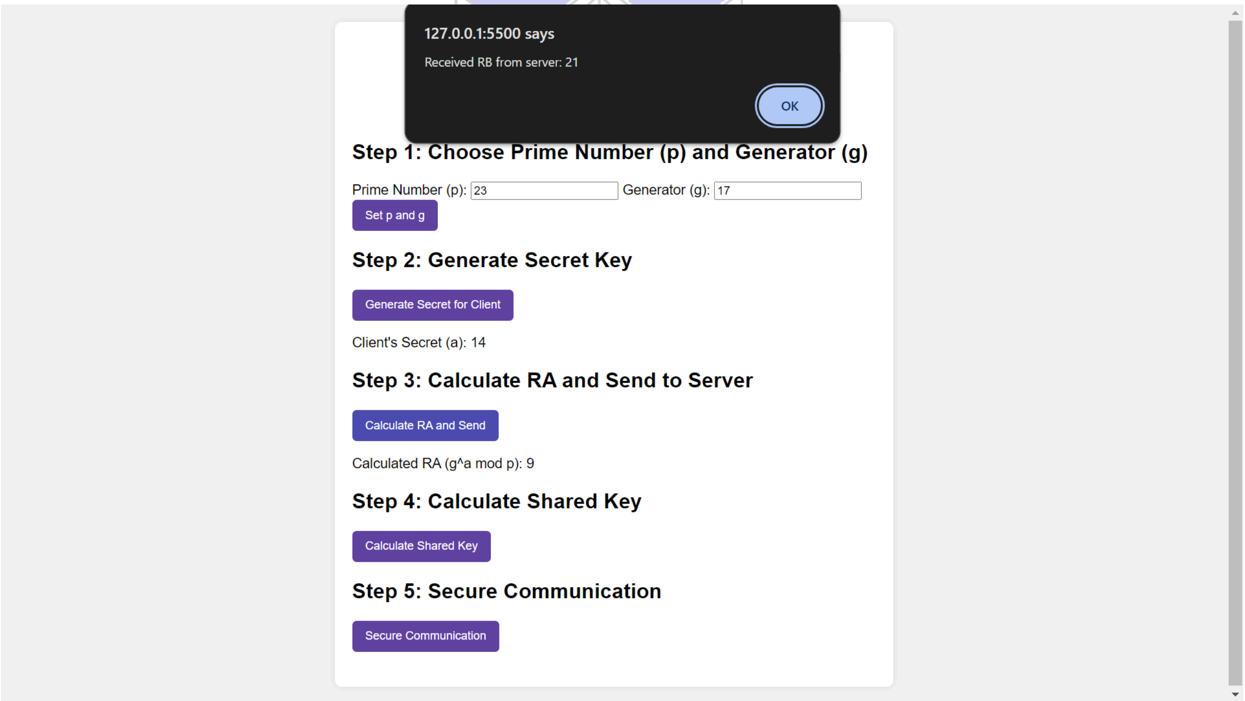
```
                document.getElementById('decrypted-message').innerText =
`Decrypted Message: ${decryptedMessage}`;
            });
        }
    </script>
</body>
</html>
```

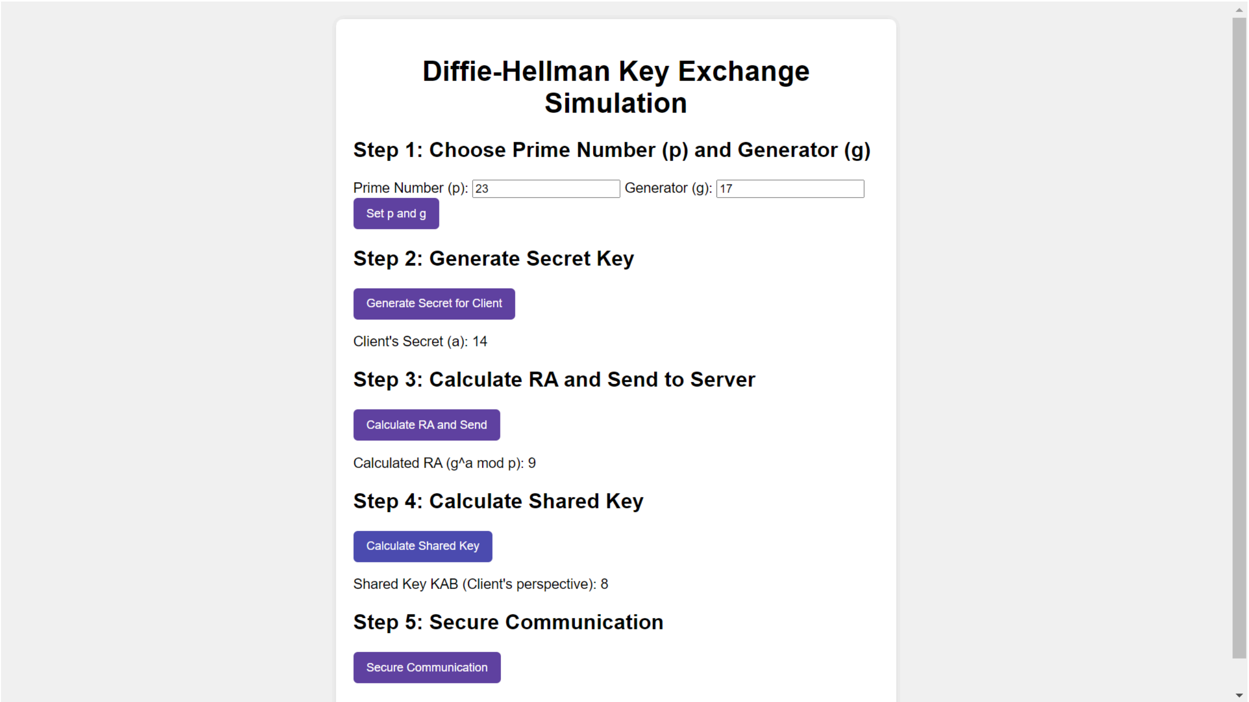
Output Snapshots:

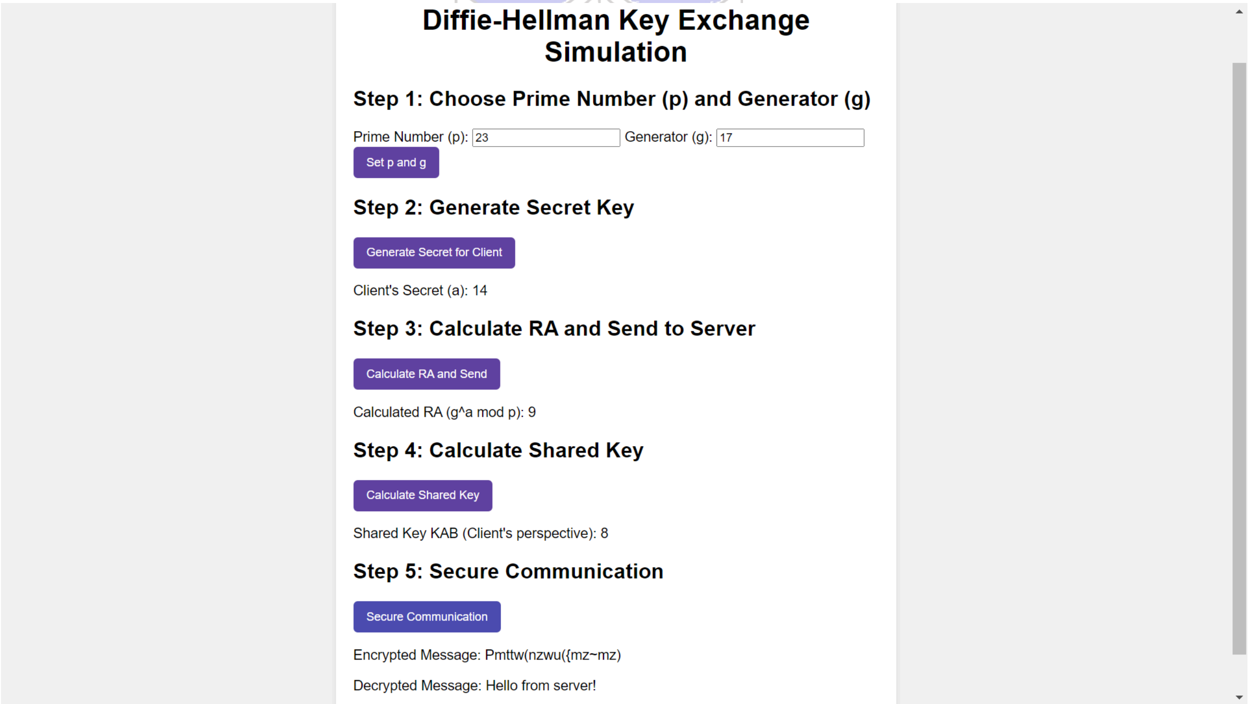





Diffie-Hellman Key Exchange Simulation
Step 1: Choose Prime Number (p) and Generator (g)
Prime Number (p): 23 Generator (g): 17
Set p and g
Step 2: Generate Secret Key
Generate Secret for Client
Client's Secret (a): 14
Step 3: Calculate RA and Send to Server
Calculate RA and Send
Step 4: Calculate Shared Key
Calculate Shared Key
Step 5: Secure Communication
Secure Communication


127.0.0.1:5500 says
Received RB from server: 21
OK
Step 1: Choose Prime Number (p) and Generator (g)
Prime Number (p): 23 Generator (g): 17
Set p and g
Step 2: Generate Secret Key
Generate Secret for Client
Client's Secret (a): 14
Step 3: Calculate RA and Send to Server
Calculate RA and Send
Calculated RA (g^a mod p): 9
Step 4: Calculate Shared Key
Calculate Shared Key
Step 5: Secure Communication
Secure Communication


Diffie-Hellman Key Exchange Simulation
Step 1: Choose Prime Number (p) and Generator (g)
Prime Number (p): 23 Generator (g): 17
Set p and g
Step 2: Generate Secret Key
Generate Secret for Client
Client's Secret (a): 14
Step 3: Calculate RA and Send to Server
Calculate RA and Send
Calculated RA (g^a mod p): 9
Step 4: Calculate Shared Key
Calculate Shared Key
Shared Key KAB (Client's perspective): 8
Step 5: Secure Communication
Secure Communication


Diffie-Hellman Key Exchange Simulation
Step 1: Choose Prime Number (p) and Generator (g)
Prime Number (p): 23 Generator (g): 17
Set p and g
Step 2: Generate Secret Key
Generate Secret for Client
Client's Secret (a): 14
Step 3: Calculate RA and Send to Server
Calculate RA and Send
Calculated RA (g^a mod p): 9
Step 4: Calculate Shared Key
Calculate Shared Key
Shared Key KAB (Client's perspective): 8
Step 5: Secure Communication
Secure Communication
Encrypted Message: Pmttw(nzwu({mz~mz)
Decrypted Message: Hello from server!

```
Command Prompt - python3
Microsoft Windows [Version 10.0.22631.4037]
(c) Microsoft Corporation. All rights reserved.

C:\Users\chand>cd Downloads\V SEM\INS\EXP6

C:\Users\chand\Downloads\V SEM\INS\EXP6>python3 server.py
Server is starting.
 * Serving Flask app 'server'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5500
Press CTRL+C to quit
 * Restarting with stat
Server is starting.
 * Debugger is active!
 * Debugger PIN: 236-475-935
127.0.0.1 - - [25/Aug/2024 17:04:57] "GET / HTTP/1.1" 200 -
Received from client - p: 23, g: 17, RA: 9
Server's secret key (b): 5
Calculated RB (g^b mod p): 21
Calculated shared key KAB: 8
127.0.0.1 - - [25/Aug/2024 17:05:54] "POST /exchange HTTP/1.1" 200 -
Received shared key from client: 8
Encrypted message to send to client: Pmttw(nzwu({mz~mz)
Decrypted message on server-side: Hello from server!
127.0.0.1 - - [25/Aug/2024 17:06:16] "POST /secure_communication HTTP/1.1" 200 -
```

Questions:

1. **Explain any one attack on Diffie-Hellman key exchange protocol.**
   Ans: Man-in-the-Middle Attack – In a Man-in-the-Middle (MitM) attack, an attacker intercepts the communication between Alice and Bob. The attacker, Mallory, can intercept the public keys exchanged and substitute their own public keys. Alice and Bob think they are securely exchanging keys, but Mallory can decrypt the messages sent by both parties, read or alter them, and then re-encrypt and forward the messages to the intended recipient. This attack compromises the security of the key exchange since the attacker ends up with the same shared secret key as both Alice and Bob. To prevent this attack, parties should use authentication mechanisms to verify each other's identity before exchanging keys.

Outcomes: Illustrate different cryptographic algorithms for security

---

Conclusion:

This implementation demonstrates the Diffie-Hellman key exchange protocol, allowing secure communication using an additive cipher. It illustrates how public and private keys can be used to derive a shared secret over an insecure channel. However, real-world applications should employ more sophisticated encryption and authentication methods to ensure security against various attacks.
