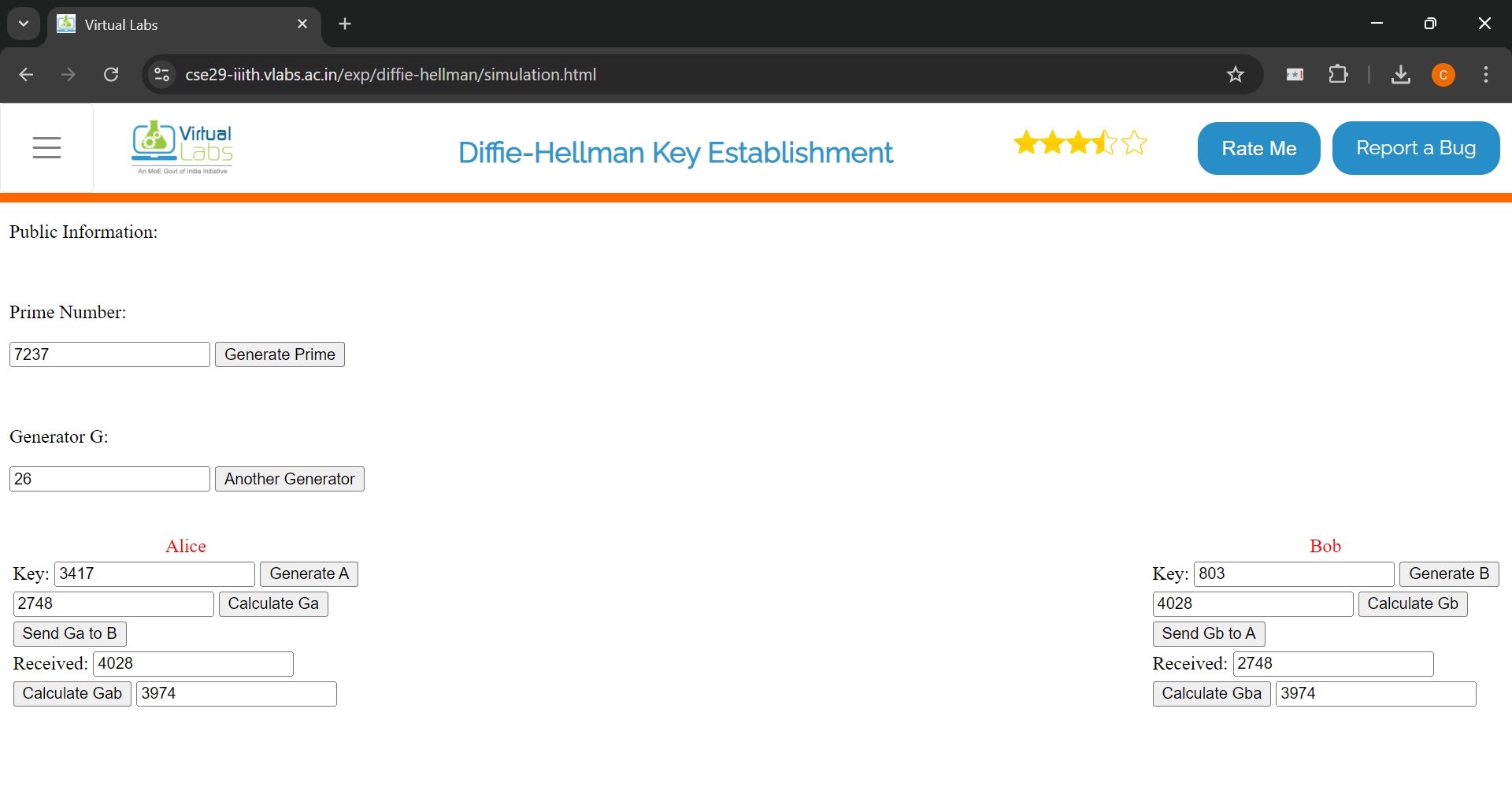


**Batch: B-3 Roll No.: 16010422234 Experiment No.: 06**

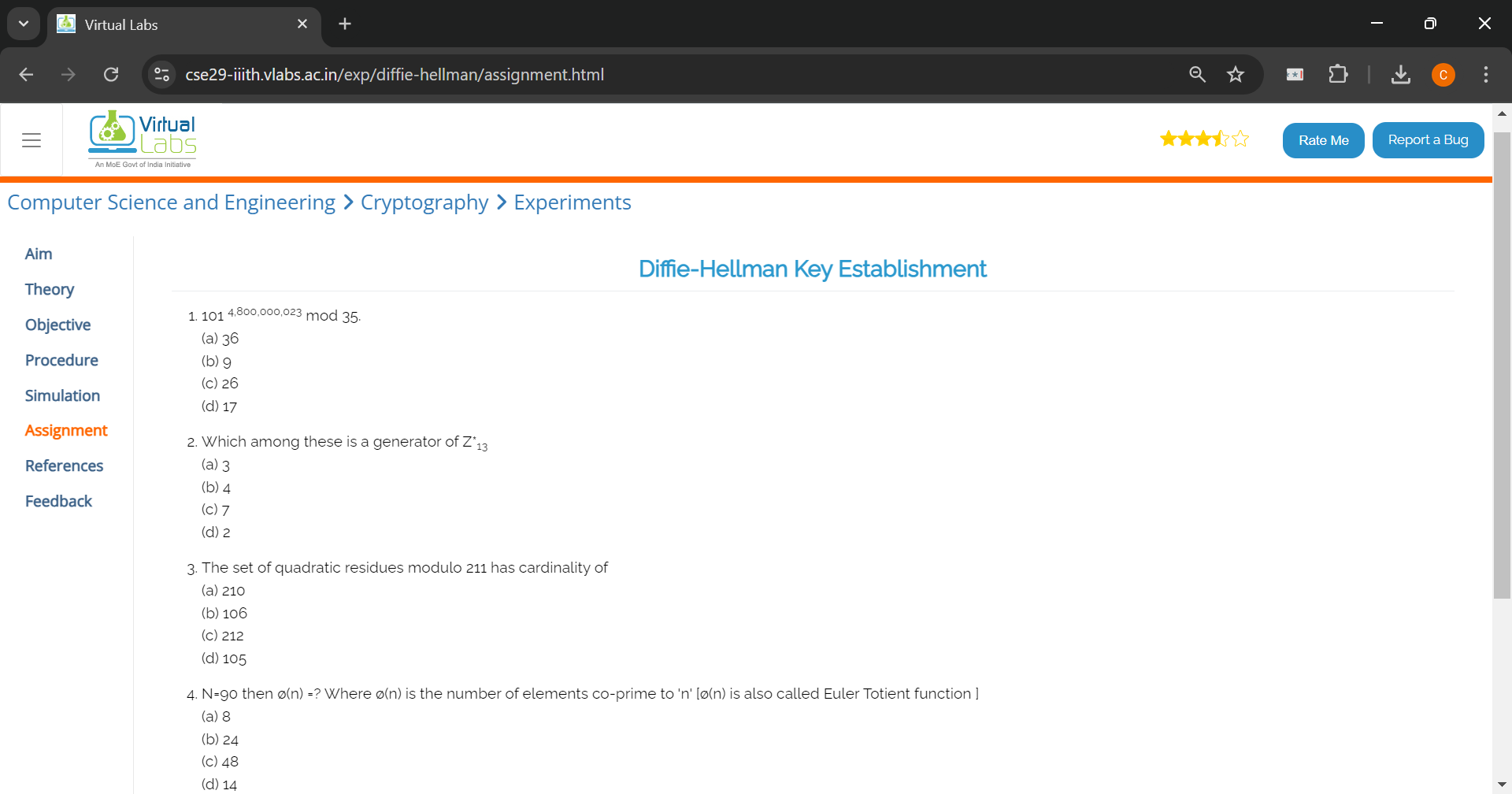
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Results:**

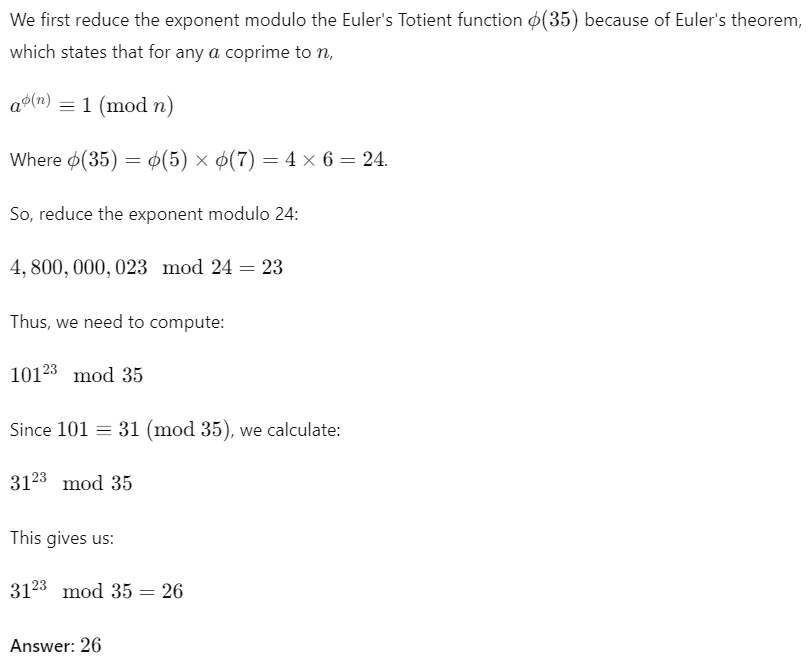
**Simulation:**



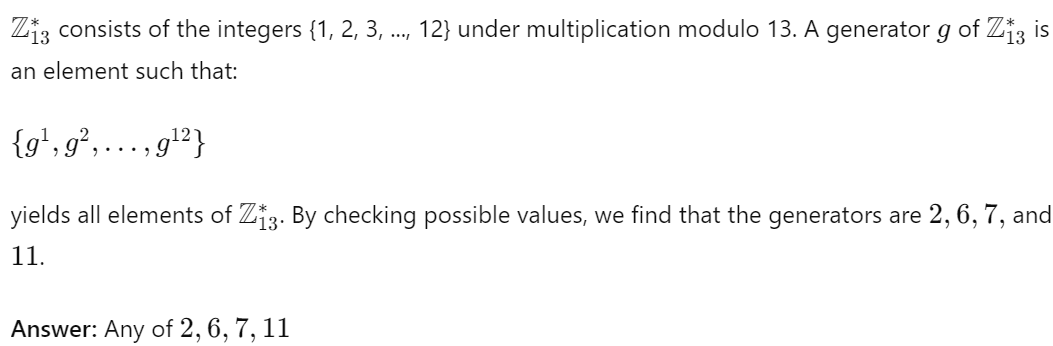
**Assignment:**



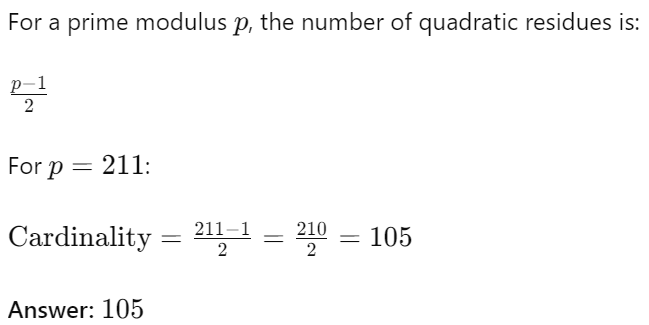
**Q1. (c) 26**



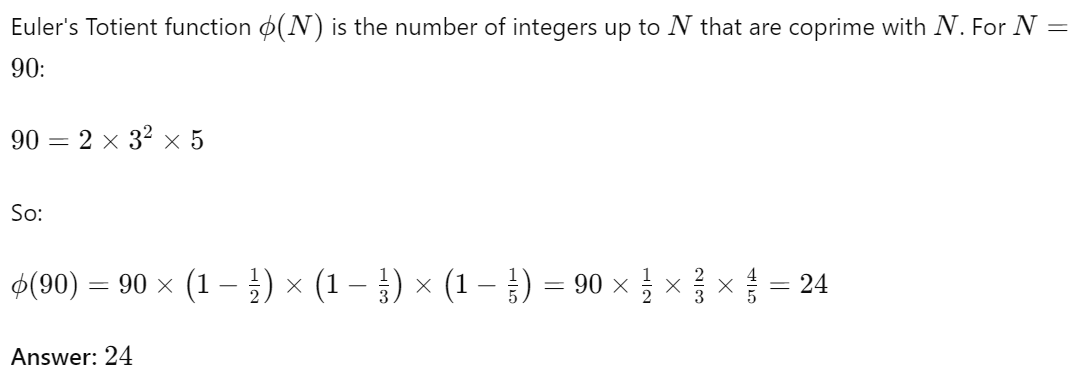
**Q2. (c), (d)**



**Q3. (d)**



**Q4. (b)**



**server.py**

**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}`;**

**});**

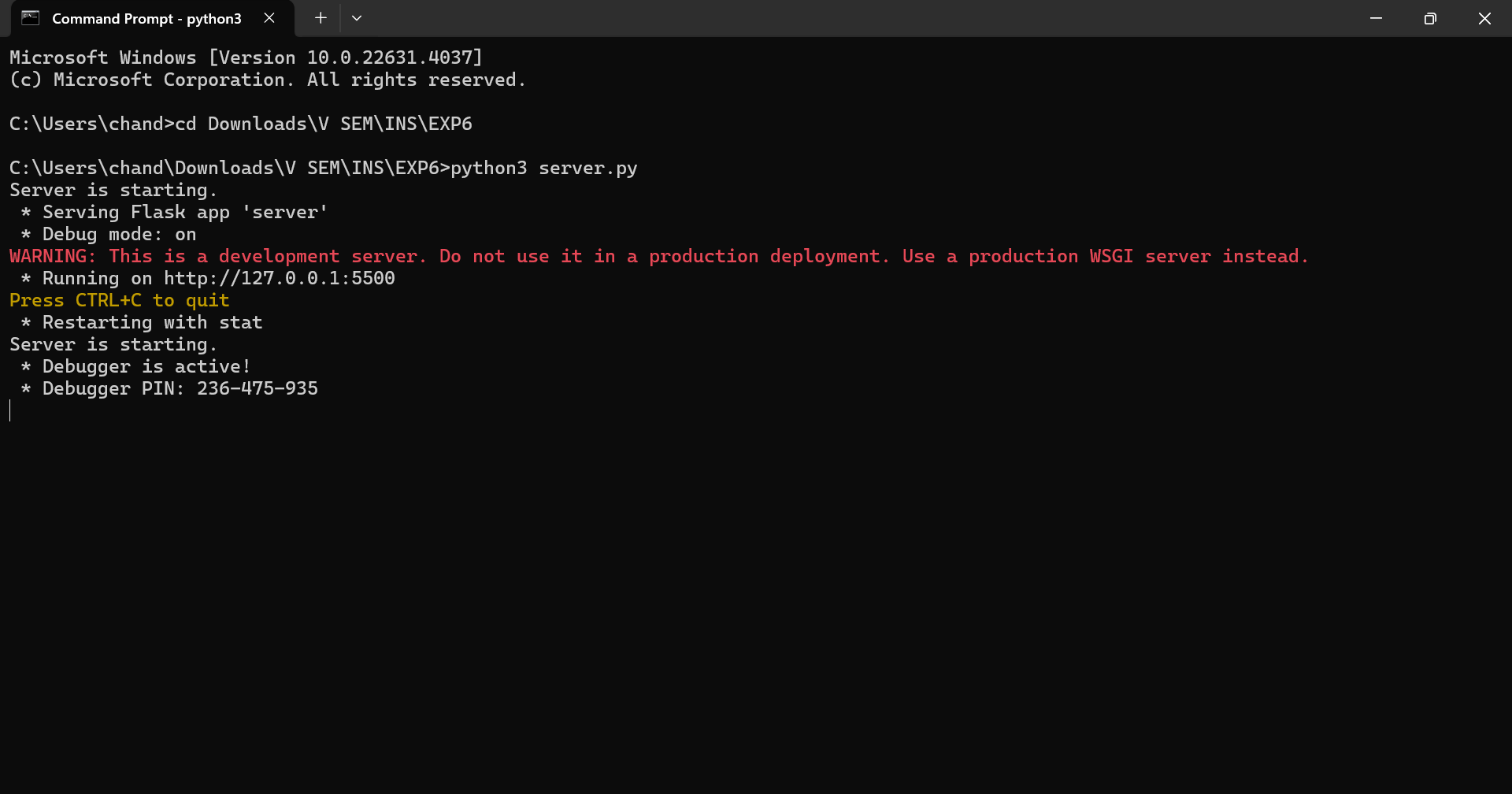
**}**

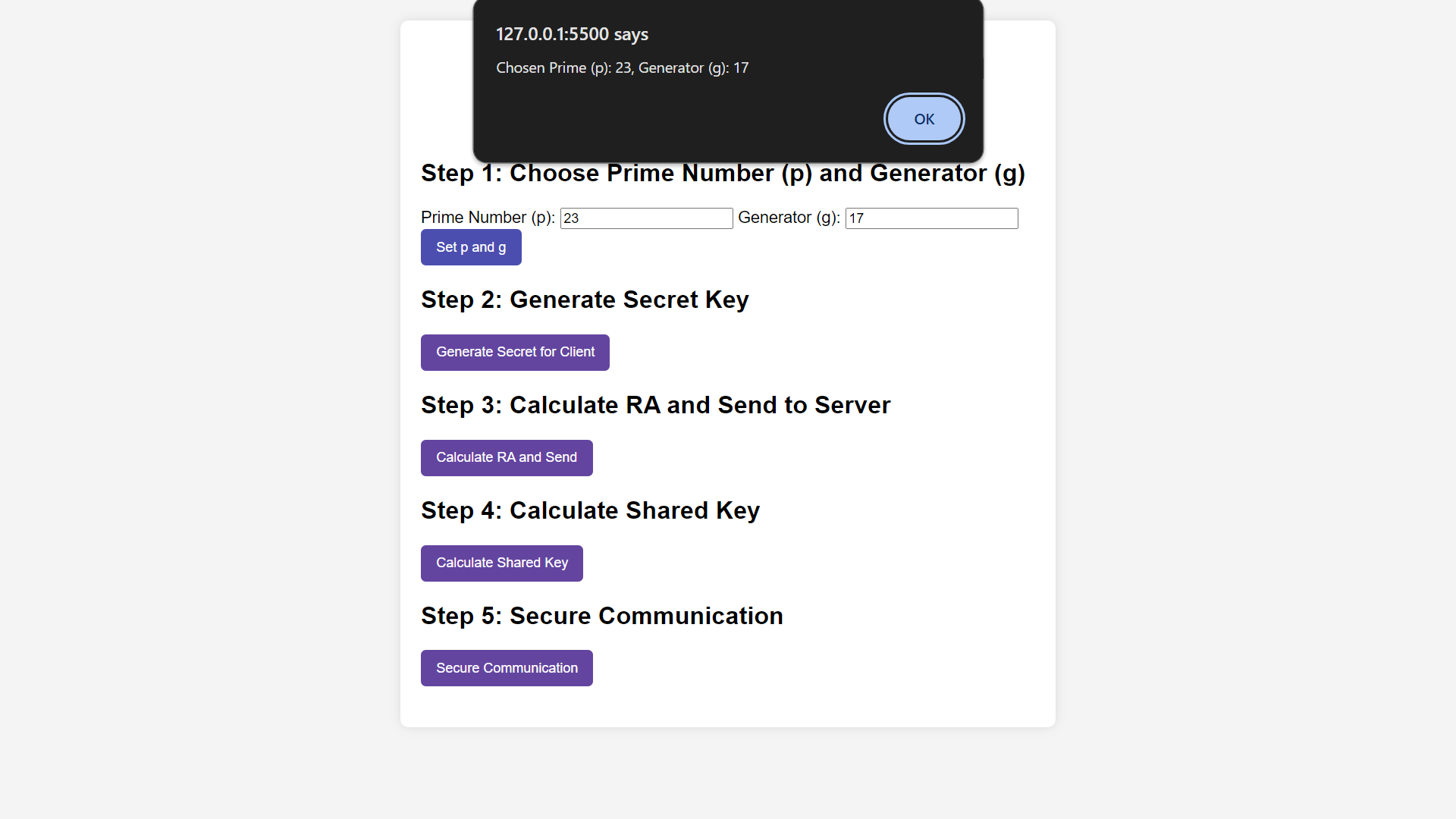
**</script>**

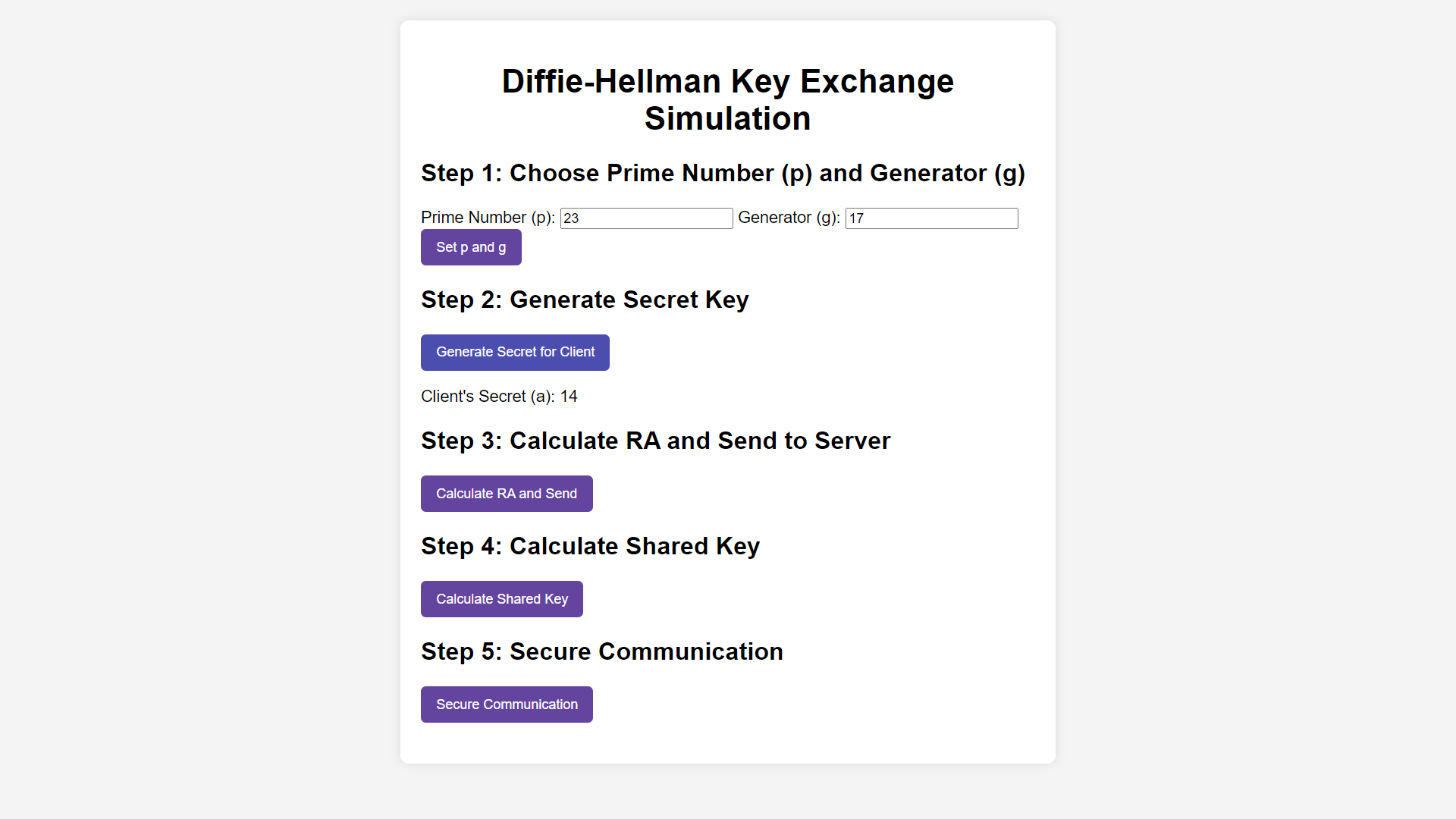
**</body>**

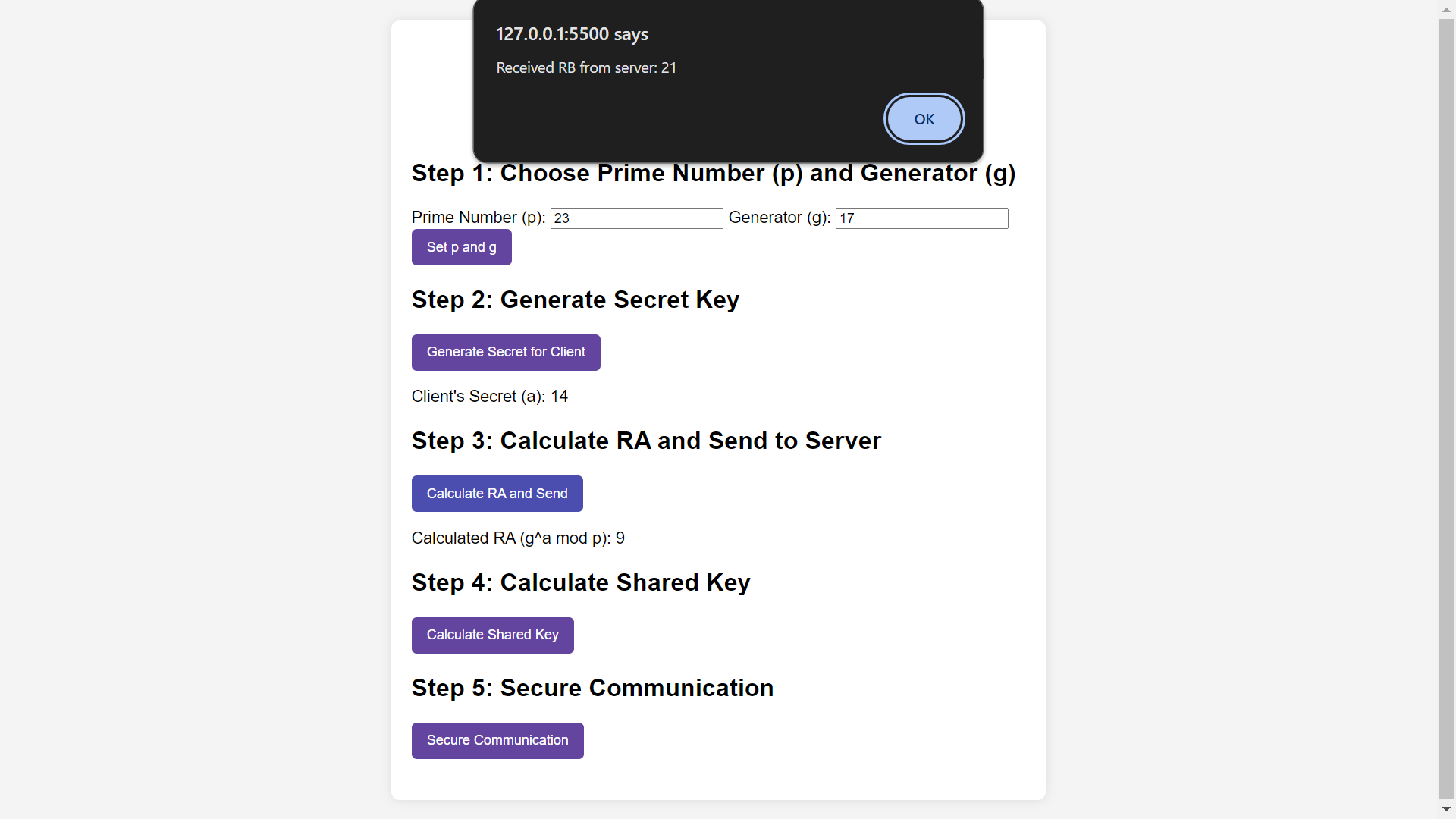
**</html>**

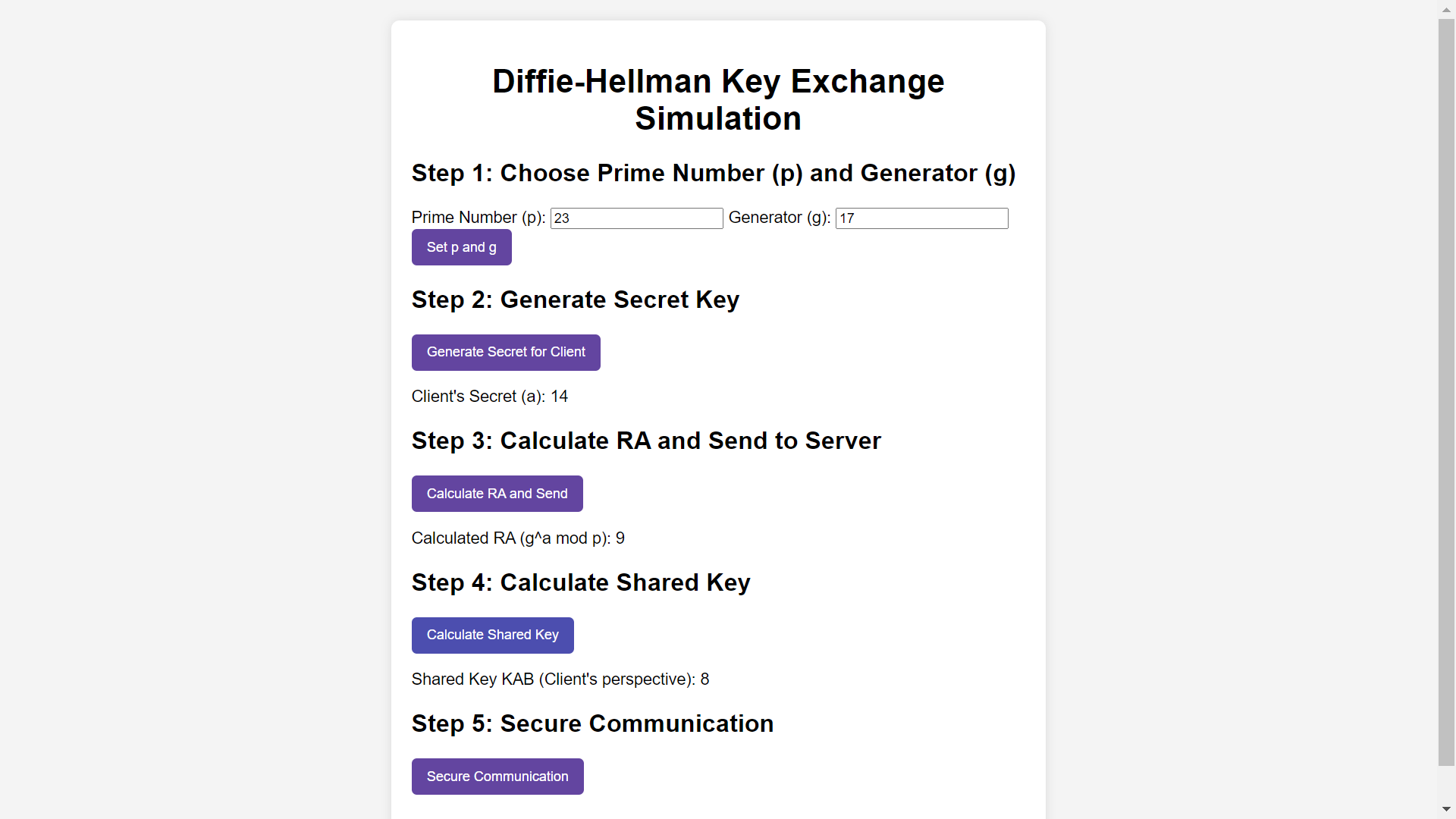
**Output Snapshots:**

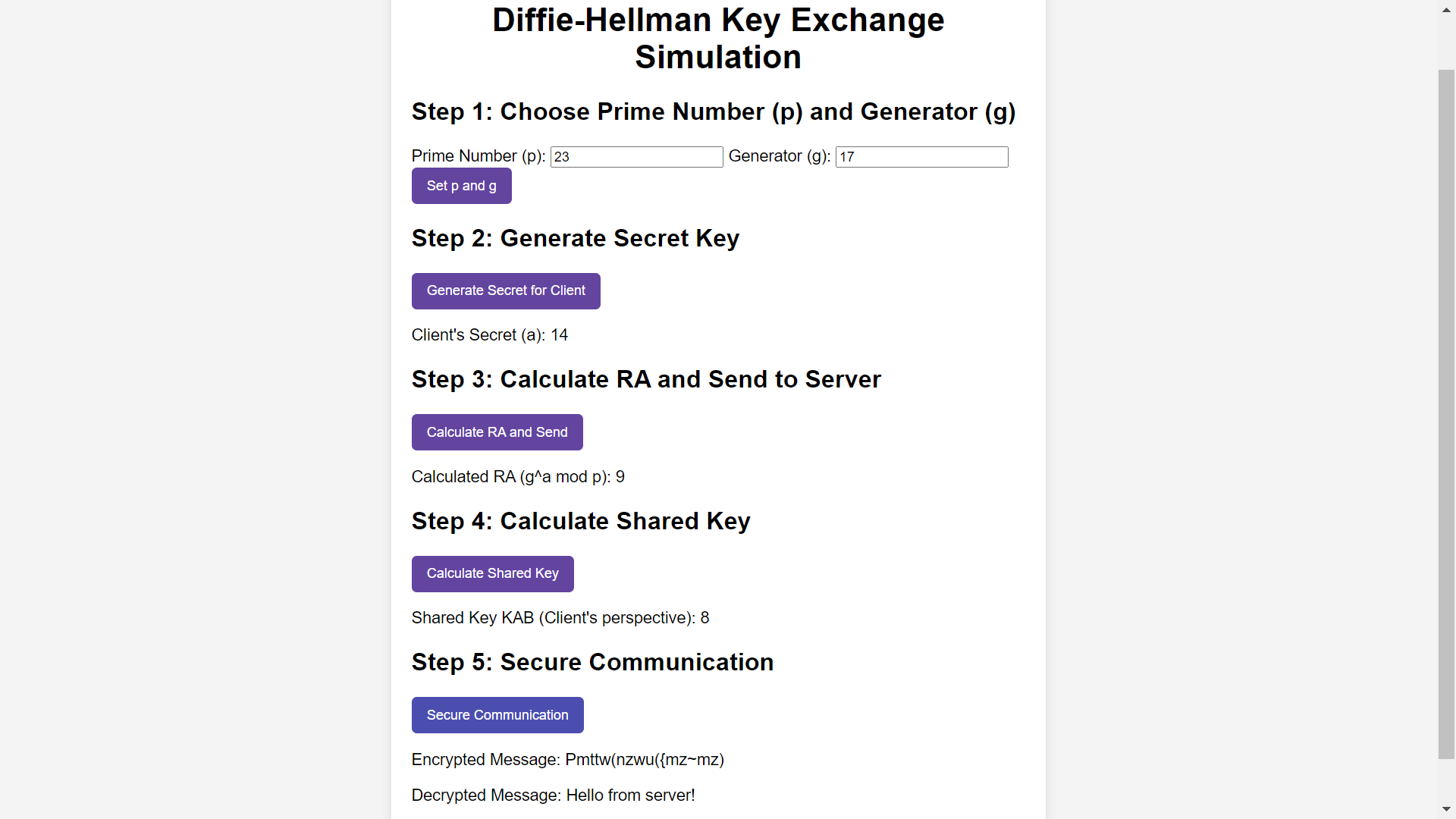


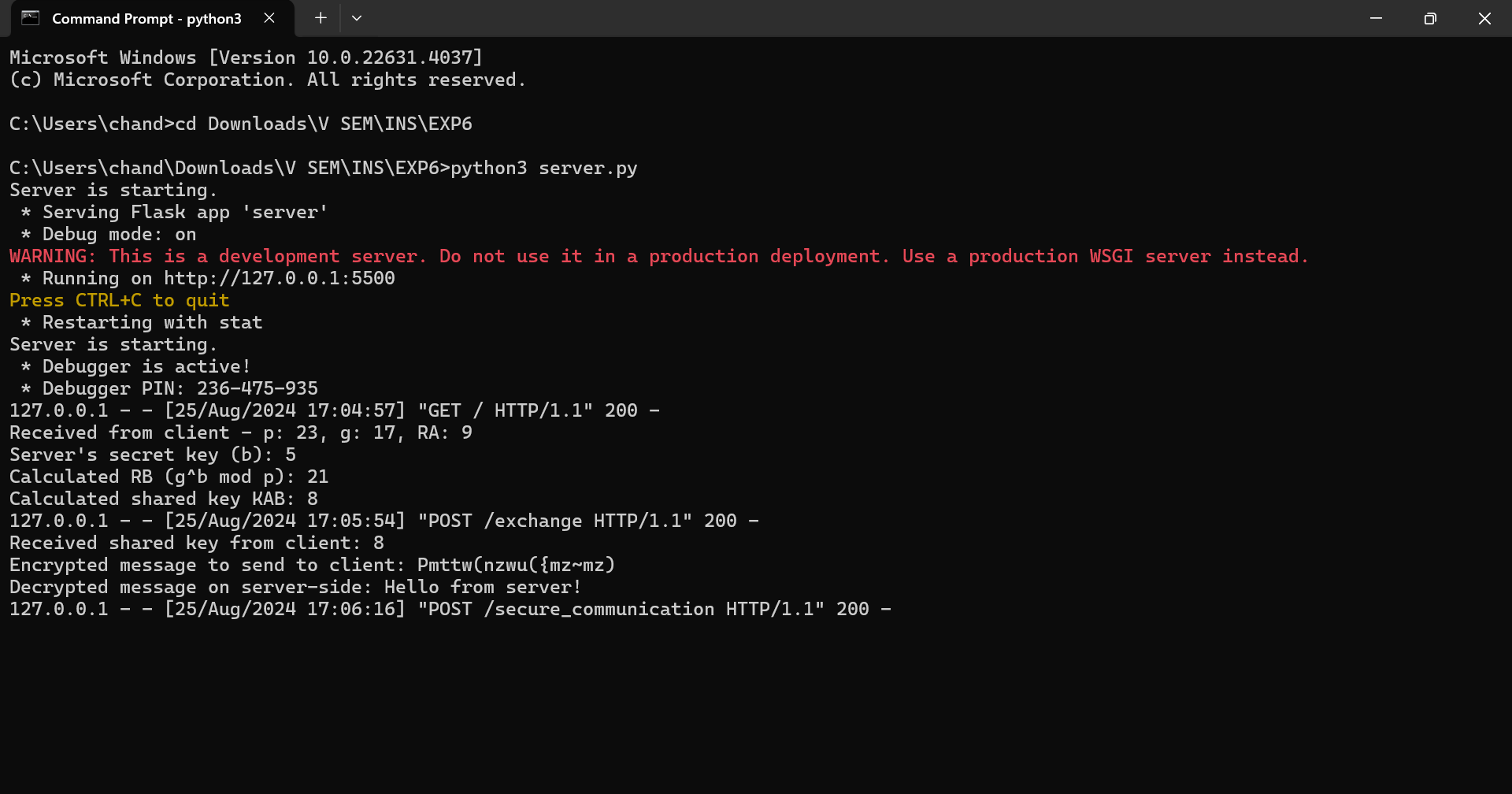












**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**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.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**