Documentation

This document provides a technical overview of how the **Secure Bluetooth Device Manager System** operates, including device discovery, whitelisting, encryption mechanisms, and communication logic.

Device Identification Logic

The system uses the bleak library to scan for real nearby Bluetooth Low Energy (BLE) devices. Each device discovered is classified and stored with the following metadata:

- **Name**: The advertised name of the device (if available).
- **MAC Address**: A unique identifier for the device.
- **Type**: Currently set to BLE for all scanned devices (simulated classification for Classic devices is reserved for extension).

The devices are displayed in the following format:

Name: OPPO Enco Buds 2 | Type: BLE | MAC: A1:B2:C3:D4:E5:F6

The identification logic ensures that the user has enough information to determine which devices to trust or whitelist.

Whitelisting Mechanism

The system implements a simple whitelisting model to restrict communication to only trusted devices.

- **Whitelisted devices** are stored as a set of MAC addresses.
- The whitelist is managed through a CLI interface that allows the user to:
    - Add a device to the whitelist by MAC
    - Remove a device from the whitelist
    - View all currently whitelisted devices

This acts as an access control mechanism, ensuring that secure communication is only initiated with approved devices.

Encryption and Handshake Process

The system establishes a secure channel using **AES (Advanced Encryption Standard)** in **CBC (Cipher Block Chaining)** mode. Encryption and decryption are handled by a SecureChannel class, built with pycryptodome.

Details:

- **Key Generation**: A 128-bit AES key is generated for each session.
- **IV (Initialization Vector)**: A unique IV is generated alongside the key.
- **Padding**: PKCS7 is used to align plaintext to AES block size.
- **Handshake**: Simulated. The key and IV are shared between peers (server and client) at runtime. No actual key exchange protocol (e.g., Diffie-Hellman) is implemented, as this is a controlled simulation.

This model ensures message confidentiality for both local and socket-based communications.

Simulated Secure Communication

Once a device is whitelisted, the user can initiate secure communication:

1. A secure channel is established using a shared AES key and IV.
2. The user's message is encrypted and printed in its ciphertext form.
3. A simulated encrypted response is sent and decrypted for display.

Example Session:
Secure session with OPPO Enco Buds 2 established.
Encrypted message sent: 8a72fc3e098f1a...
Encrypted message received: 3be44a59a82e2d...
Decrypted response: Ack from OPPO Enco Buds 2

Socket-Based Secure Communication (Bonus)

The project includes a simulation of secure TCP-based communication:

- A local **server** is started on 127.0.0.1:9090.
- The **client** connects and sends an AES-encrypted message.
- The server decrypts the message, processes it, and replies with an encrypted response.

Results :

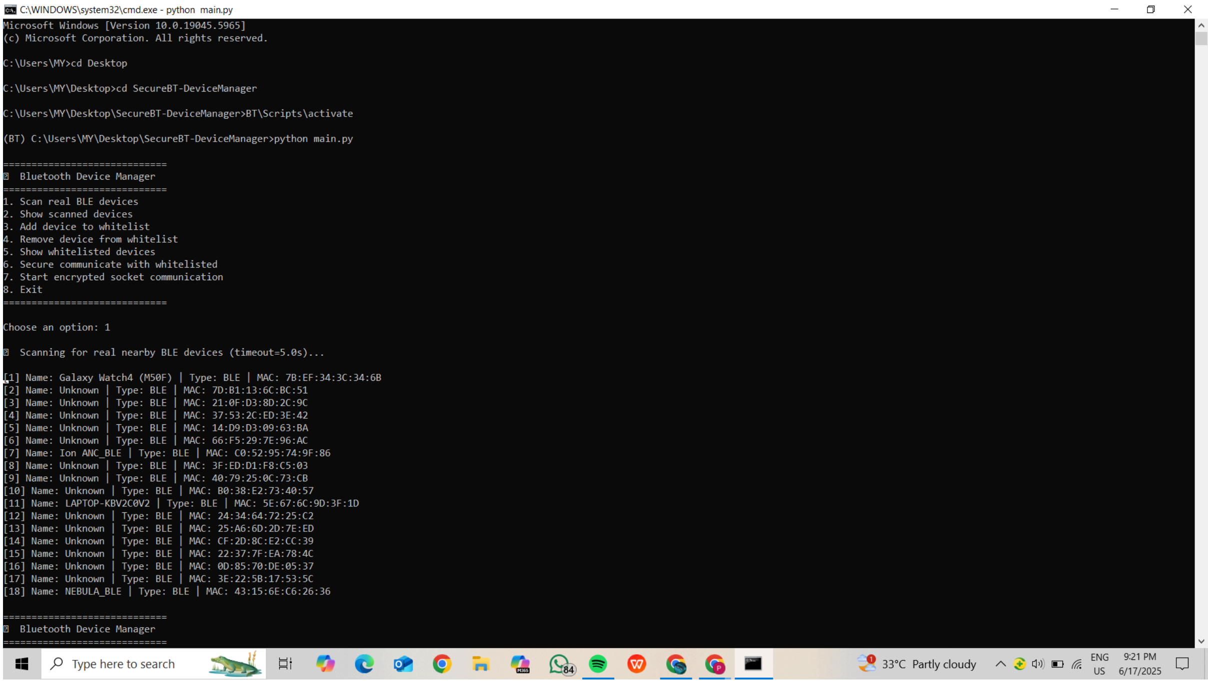


Image :1 [Shows Scanning of all the nearby devices.

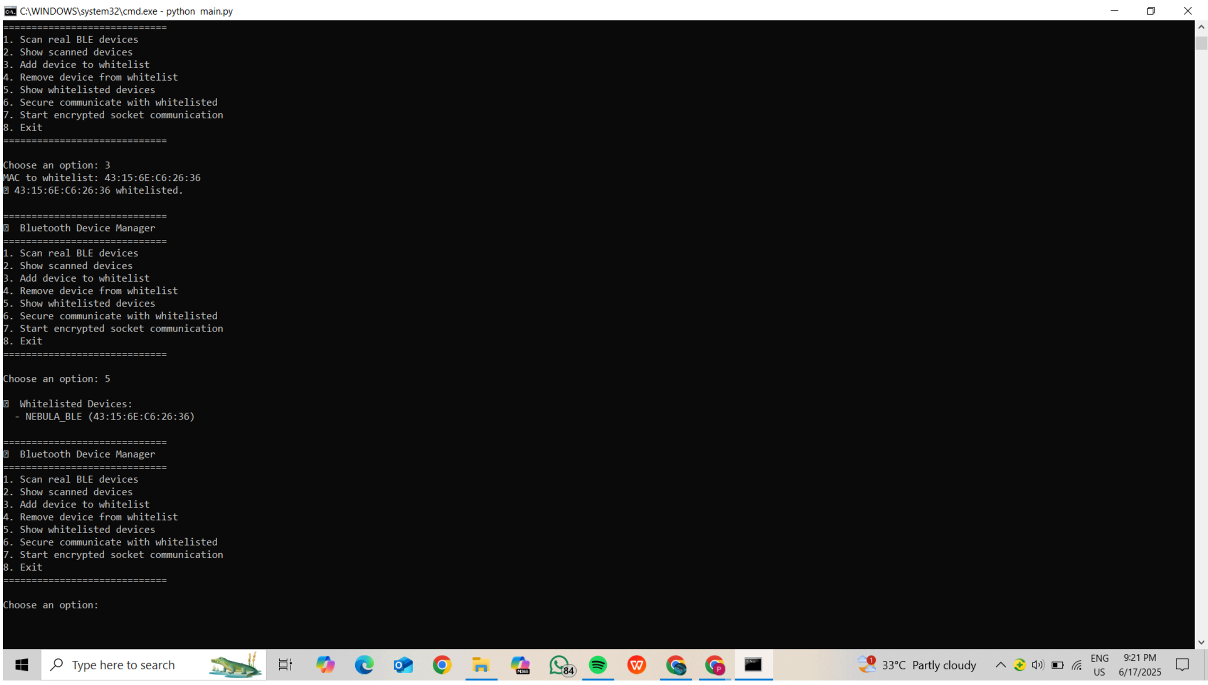


Image :2 [Shows how Device is Whislisted]

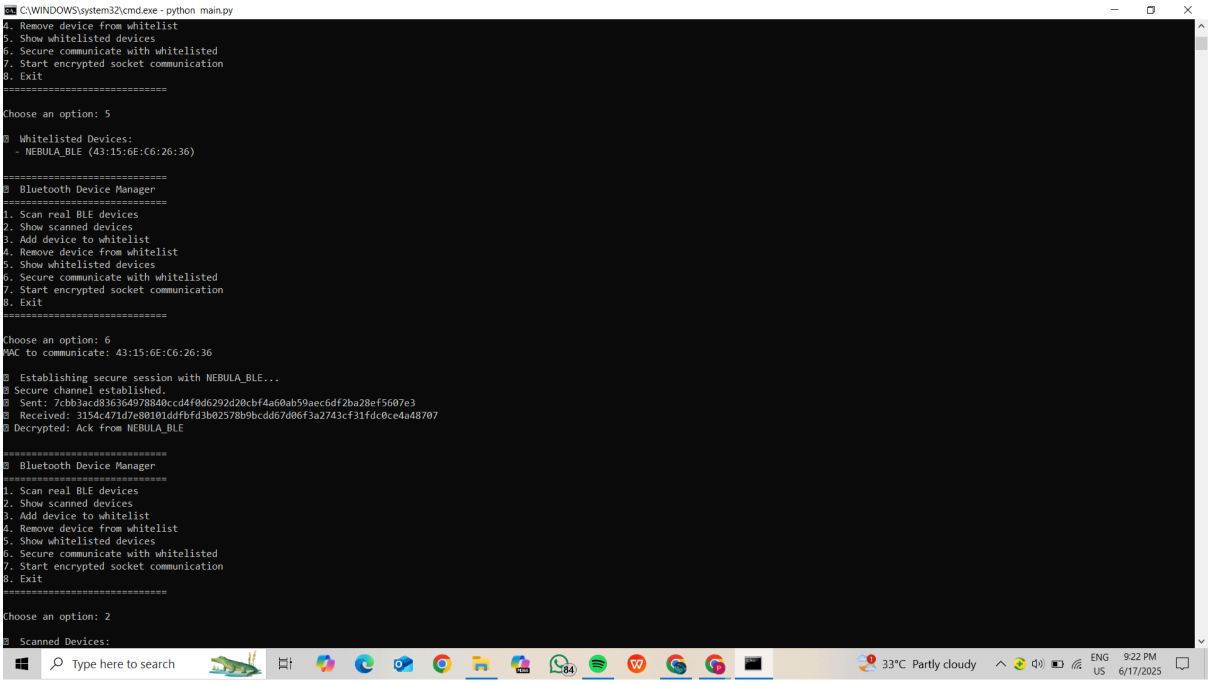


Image :3 [Shows device is securely communicated using AES]

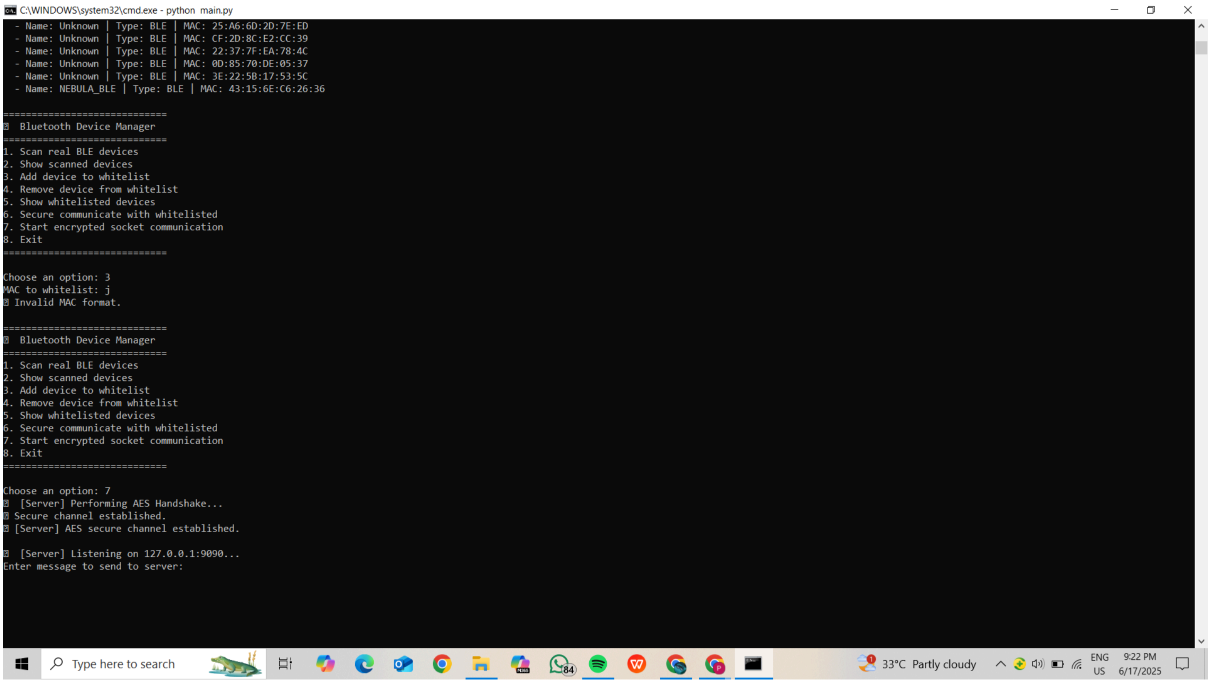
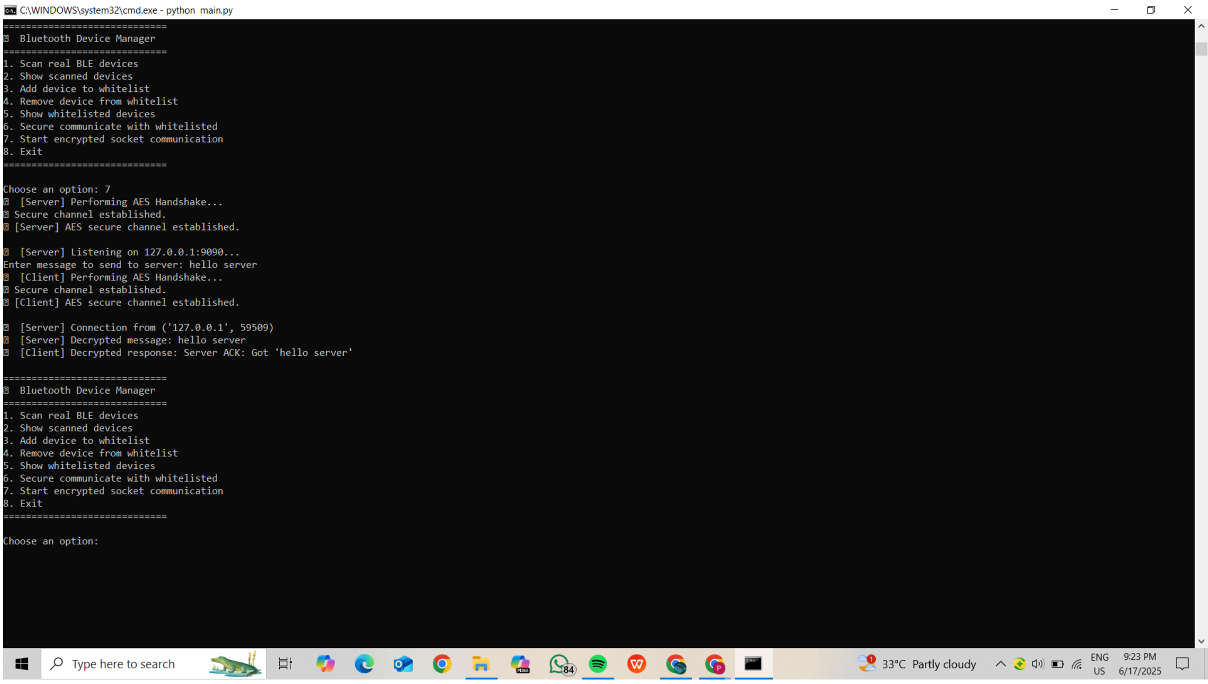


Image: 4 [Shows started Secure encrypted Socket Communication]



Example:

[Server] Secure channel established.
[Server] Listening on 127.0.0.1:9090
[Client] Secure channel established.
[Client] Message sent: Hello Server
[Server] Decrypted message: Hello Server
[Client] Decrypted response: Server ACK: Got 'Hello Server'

This simulates encrypted data exchange over a potentially insecure network using pre-shared secrets.

Conclusion

The system is modular, secure, and designed to operate with real BLE hardware (when available) while offering full simulation capabilities for environments without Bluetooth access. It provides a foundation for further development in secure IoT device management or Bluetooth-based access control systems.