```
#!/usr/bin/env python3
import os
import sys
import json
import socket
import threading
import getpass
import time
import subprocess
import urllib.request
from cryptography.fernet import Fernet

REQUIRED_MODULES = ["cryptography", "pyperclip", "miniupnpc"]

def ensure_dependencies():
    """Ensure required dependencies are installed."""
    for module in REQUIRED_MODULES:
        try:
            __import__(module)
        except ImportError:
            print(f"[INFO] Module '{module}' not found. Installing...")
            subprocess.check_call([sys.executable, "-m", "pip", "install", module])

ensure_dependencies()

def get_public_ip(timeout=5):
    """Retrieve the host's public IP address using api.ipify.org."""
    try:
```

```
        with urllib.request.urlopen("https://api.ipify.org", timeout=timeout) as response:
            ip = response.read().decode("utf8")
            return ip
    except Exception as e:
        print(f"[WARN] Failed to retrieve public IP: {e}")
        return None


def setup_nat(port, description="AGC Secure Session"):
    """
    Attempt to map the given TCP port using UPnP.
    Returns (external_ip, mapping_success) if UPnP is successful; otherwise (None, False).
    """
    try:
        import miniupnpc
        upnpc = miniupnpc.UPnP()
        upnpc.discoverdelay = 200
        ndevices = upnpc.discover()
        if ndevices > 0:
            upnpc.selectigd()
            external_ip = upnpc.externalipaddress()
            # Use None instead of an empty string to avoid "Invalid Args" error on some routers
            mapping = upnpc.addportmapping(port, 'TCP', upnpc.lanaddr, port, description,
None)
            if mapping:
                print(f"[INFO] Port {port} successfully mapped via UPnP. External IP: {external_ip}")
                return external_ip, True
            else:
                print("[WARN] UPnP gateway found but port mapping failed.")
```

```
            return external_ip, False
        else:
            print("[WARN] No UPnP-enabled router found.")
            return None, False
    except Exception as e:
        print(f"[WARN] NAT traversal using UPnP failed: {e}")
        return None, False

def generate_session_key():
    """Generate an encryption key for the session."""
    return Fernet.generate_key()

def load_fernet(session_key):
    """Initialize and return a Fernet object with the session key."""
    return Fernet(session_key)

def encrypt_message(fernet, message: bytes) -> bytes:
    """Encrypt a message (in bytes)."""
    return fernet.encrypt(message)

def decrypt_message(fernet, token: bytes) -> bytes:
    """Decrypt the token. Returns an error message if decryption fails."""
    try:
        return fernet.decrypt(token)
    except Exception:
        return b"[ERROR: Unable to decrypt message]"

SETTINGS_FILE = "chat_settings.json"
```

```
CHAT_HISTORY_FILE = "chat_history.log"


def load_settings():
    """Load settings from a JSON file; return defaults if not found."""
    if os.path.exists(SETTINGS_FILE):
        with open(SETTINGS_FILE, "r") as f:
            return json.load(f)
    return {"chat_history": True, "contacts": {}}


def save_settings(settings):
    """Save settings to a JSON file."""
    with open(SETTINGS_FILE, "w") as f:
        json.dump(settings, f, indent=4)


def log_chat(message: str):
    """Append a message to the chat log if enabled."""
    settings = load_settings()
    if settings.get("chat_history", True):
        with open(CHAT_HISTORY_FILE, "a") as f:
            f.write(message + "\n")


def send_file(fernet, conn, filename):
    """Send a file securely to the peer."""
    if not os.path.exists(filename):
        print(f"[ERROR] File '{filename}' not found.")
        return
    try:
        with open(filename, "rb") as f:
```

```
            content = f.read()
        payload = b"[FILE]" + filename.encode() + b"::" + content
        encrypted = encrypt_message(fernet, payload)
        conn.sendall(encrypted)
        print(f"[INFO] File '{filename}' sent successfully.")
        log_chat(f"Sent file: {filename}")
    except Exception as e:
        print(f"[ERROR] Failed to send file: {e}")


def handle_received_data(fernet, data):
    """Handle decrypted data as a message or file."""
    dec = decrypt_message(fernet, data)
    if dec.startswith(b"[FILE]"):
        try:
            content = dec[len(b"[FILE]"):]
            filename, file_content = content.split(b"::", 1)
            filename = filename.decode()
            received_filename = "received_" + filename
            with open(received_filename, "wb") as f:
                f.write(file_content)
            print(f"\n[INFO] Received file saved as: {received_filename}")
            log_chat(f"Received file: {filename}")
        except Exception:
            print("[ERROR] Failed to parse received file data.")
    else:
        try:
            message = dec.decode()
            print("\nPeer:", message)
```

```
            log_chat("Peer: " + message)
        except Exception:
            print("[ERROR] Unable to decode incoming message.")


def chat_listener(conn, fernet):
    """Continuously listen for incoming messages."""
    while True:
        try:
            data = conn.recv(4096)
            if not data:
                print("[INFO] Connection closed by peer.")
                break
            handle_received_data(fernet, data)
        except Exception as e:
            print(f"[ERROR] Problem receiving data: {e}")
            break


def chat_sender(conn, fernet):
    """Send messages or commands to your chat partner."""
    help_msg = (
        "\n[COMMANDS]\n"
        "  /file <path>   : Send a file\n"
        "  /delchat       : Delete chat history\n"
        "  /exit          : Exit chat\n"
    )
    print(help_msg)
    while True:
        msg = input("> ").strip()
```

```
        if not msg:
            continue
        if msg == "/exit":
            conn.close()
            print("Goodbye! Chat session ended.")
            break
        elif msg == "/delchat":
            if os.path.exists(CHAT_HISTORY_FILE):
                os.remove(CHAT_HISTORY_FILE)
                print("[INFO] Chat history removed.")
            else:
                print("[INFO] No chat history found.")
        elif msg.startswith("/file "):
            _, filename = msg.split(" ", 1)
            send_file(fernet, conn, filename.strip())
        else:
            try:
                encrypted = encrypt_message(fernet, msg.encode())
                conn.sendall(encrypted)
                log_chat("Me: " + msg)
            except Exception as e:
                print(f"[ERROR] Failed to send message: {e}")
                break

def run_host():
    """Run in Host mode with NAT assistance and authentication."""
    HOST = ''
    DEFAULT_PORT = 5000
```

```
PORT = DEFAULT_PORT

print("\n[HOST MODE] Starting chat session...")

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

while True:
    try:
        server_socket.bind((HOST, PORT))
        break
    except Exception as e:
        print(f"[ERROR] Unable to bind port {PORT}: {e}")
        new_port = input("Enter a different port number: ").strip()
        try:
            PORT = int(new_port)
        except ValueError:
            print("[ERROR] Invalid port number. Please try again.")

server_socket.listen(1)
print(f"[INFO] Listening on port {PORT}.")

settings = load_settings()
stored_pass = settings.get("password")
if not stored_pass:
    stored_pass = getpass.getpass("Set a session password: ")
    settings["password"] = stored_pass
    save_settings(settings)
```

```
nat_ip, mapping_success = setup_nat(PORT)
if mapping_success and nat_ip:
    host_ip = nat_ip
    print(f"[INFO] Using NAT-mapped external IP: {host_ip}")
else:
    host_ip = get_public_ip()
    if not host_ip:
        host_ip = socket.gethostbyname(socket.gethostname())
        print(f"[WARN] Could not retrieve public IP. Using local IP: {host_ip}")
    else:
        print(f"[INFO] Public IP retrieved: {host_ip}")

connection_info = (
    f"\n[HOST INFO]\n"
    f"IP Address: {host_ip}\n"
    f"Port: {PORT}\n"
    f"Session Password: {stored_pass}\n"
    "Note: Forward this port if behind NAT.\n"
)
try:
    import pyperclip
    pyperclip.copy(connection_info)
    print("[INFO] Connection details copied to clipboard.")
except Exception as e:
    print(f"[WARN] Clipboard copy failed: {e}")
print(connection_info)
print("[INFO] Waiting for a client connection...\n")
```

```
    conn, addr = server_socket.accept()
    print(f"[INFO] Connected to {addr}.")

    conn.sendall(b"[AUTH] Please send your session password.")
    peer_pass = conn.recv(1024).decode().strip()
    if peer_pass != stored_pass:
        conn.sendall(b"[AUTH_FAIL] Incorrect password. Connection refused.")
        print("[ERROR] Incorrect password entered by client. Disconnecting...")
        conn.close()
        return
    else:
        conn.sendall(b"[AUTH_OK]")
        print("[INFO] Client authenticated successfully.")

    session_key = generate_session_key()
    time.sleep(0.5)
    conn.sendall(session_key)
    fernet = load_fernet(session_key)
    print("[INFO] Secure session established. Let the chat begin!")

    threading.Thread(target=chat_listener, args=(conn, fernet), daemon=True).start()
    chat_sender(conn, fernet)

def run_client():
    """Run in Client mode with a friendly prompt for connection details."""
    print("\n[CLIENT MODE] Enter host connection details.")
    host_ip = input("Host IP Address: ").strip()
```

```
try:
    host_port = int(input("Host Port (e.g., 5000): ").strip())
except ValueError:
    print("[ERROR] Port must be a number!")
    return

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
    try:
        client_socket.connect((host_ip, host_port))
    except Exception as e:
        print(f"[ERROR] Could not connect to host: {e}")
        return

    auth_request = client_socket.recv(1024)
    if auth_request.startswith(b"[AUTH]"):
        session_pass = getpass.getpass("Enter session password: ").strip()
        client_socket.sendall(session_pass.encode())
        auth_resp = client_socket.recv(1024)
        if auth_resp.startswith(b"[AUTH_FAIL]"):
            print("[ERROR] Authentication failed. Check password and try again.")
            return
        elif auth_resp.startswith(b"[AUTH_OK]"):
            print("[INFO] Authentication successful!")
    else:
        print("[ERROR] Unexpected authentication response.")
        return

    session_key = client_socket.recv(1024)
```

```
        fernet = load_fernet(session_key)
        print("[INFO] Secure session established with host. You may now chat.")

        threading.Thread(target=chat_listener, args=(client_socket, fernet),
daemon=True).start()
        chat_sender(client_socket, fernet)


def print_banner():
    """Display a simple welcome banner."""
    print("\nWelcome to AGC\n")


def main():
    """Main entry point: choose Host or Client mode."""
    print_banner()
    print("Select an option:")
    print("  1. Host a chat session")
    print("  2. Connect to a chat session (Client Mode)")
    choice = input("\nEnter 1 or 2: ").strip()
    if choice == "1":
        run_host()
    elif choice == "2":
        run_client()
    else:
        print("[ERROR] Invalid choice. Please restart and select 1 or 2.")


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