Mock Discovery Agent PQC Handshake Report

This example consists of 2 files, a client and a server. The client makes a TCP connection with the server and initiates a handshake to securely establish a session key and send mock system information to the server. The purpose of this program is to showcase the process of ML-KEM and ML-DSA functions when simulating a PQC supported, TLS-like handshake from a client discovery agent to a server.

Kyber-py and Dilithium-py packages (not tested for production use) were used with ML-KEM_768 and ML-DSA_65 parameter sets, respectively. The client and server are running on localhost.

Operation and Logic

First, run Discovery_Server.py, then Discovery_Client.py. The client file should initiate 5 simulated clients to create a connection simultaneously with the server to initiate the PQC handshakes. Once each handshake is complete, the server will display system information from the client.

Program Steps:

1. Server socket is setup on localhost on port 1026
2. Client sockets are setup on localhost and connect to port 1026 at once
3. Server handles each client one-by-one based on the first connection.
4. Once connected, the client sends its public ML-KEM key to the server.
5. The server creates the session key and correlated ciphertext from the client's public key and sends back the server's public ML-DSA key, ciphertext, and signature of the ciphertext (signed with the private ML-DSA key)
6. The client receives and extracts the session key from the ciphertext with the client's private ML-KEM key ONLY when the ciphertext's signature is verified with the server's public ML-DSA key. Both parties now have the same session key.
7. The client uses the session key to AES encrypt their system info (JSON format) and sends it to the server, along with their client public ML-DSA key and a signature of the system info (signed with the private ML-DSA key).
8. Now the server verifies the system info's signature with the supplied public ML-DSA key, then uses the session key to decrypt and display the system info.

(More concurrent clients can be simulated by creating more threads in Discovery_Client.py)

```
Connecting to localhost server on port 1026...
Client ('127.0.0.1', 8639) connected.
Client ('127.0.0.1', 8640) connected.
Client ('127.0.0.1', 8642) connected.
Client ('127.0.0.1', 8643) connected.
Client ('127.0.0.1', 8644) connected.

Client ('127.0.0.1', 8639) verified signature from server and retrieved secret. Encrypting and sending system info: {"hostname": "mockdevice_1", "os": "Windows 11", "ip": "127.0.0.1"}
Client ('127.0.0.1', 8639) disconnected.


Client ('127.0.0.1', 8640) verified signature from server and retrieved secret. Encrypting and sending system info: {"hostname": "mockdevice_2", "os": "Windows 10", "ip": "127.0.0.1"}
Client ('127.0.0.1', 8640) disconnected.
```

```
Server ('127.0.0.1', 1026) up.

Client ('127.0.0.1', 8639) connected.
Client ('127.0.0.1', 8639) successful handshake. System info: {"hostname": "mockdevice_1", "os": "Windows 11", "ip": "127.0.0.1"}
Client ('127.0.0.1', 8639) disconnected.


Client ('127.0.0.1', 8640) connected.
Client ('127.0.0.1', 8640) successful handshake. System info: {"hostname": "mockdevice_2", "os": "Windows 10", "ip": "127.0.0.1"}
Client ('127.0.0.1', 8640) disconnected.


Client ('127.0.0.1', 8642) connected.
Client ('127.0.0.1', 8642) successful handshake. System info: {"hostname": "mockdevice_3", "os": "Windows 11", "ip": "127.0.0.1"}
```