

# Asset Discovery Using Post Quantum Cryptography Demo Report

## Objective:

To simulate how asset discovery agents can be protected against quantum threats by replacing classical key exchange and authentication methods (e.g., RSA, ECC) with post-quantum algorithms specifically, ML-KEM (Kyber) for key encapsulation and Dilithium for digital signatures. The system ensures that before any device scanning or communication begins, a PQC-secured connection is established between the discovery agent and the remote machine.

## System Architecture

1. Target Device (Ubuntu VM)  
Acts as a simulated enterprise asset running:
  - a. SSH service (OpenSSH)
  - b. TLS service (optional OpenSSL endpoint)
  - c. SNMP agent (snmpd)
  - d. Flask API that returns a list of connected devices
  - e. PQC secure server using ML-KEM\_512 and Dilithium2
2. Host Machine (Windows/Linux/macOS)  
Runs the discovery system:
  - a. PQC utility module (pqc\_client\_connection.py) for key exchange and verification
  - b. Discovery agent script (discovery\_agent.py) that performs:
    - i. TLS check
    - ii. SSH banner grab
    - iii. SNMP query
    - iv. Retrieval of mock internal asset data All operations are blocked until the PQC handshake succeeds.

## Demo Flow

1. Setup the Ubuntu VM
  - a. Install essential services (SSH, SNMP, Flask)
  - b. Create a mock JSON file of connected devices
  - c. Start the Flask API to serve this data
  - d. Launch the PQC server which:
    - i. Receives a Kyber public key
    - ii. Sends back a ciphertext, shared secret, and Dilithium signature
2. Run Discovery from Host Machine
  - a. The agent first connects to the PQC server
  - b. After a successful ML-KEM decapsulation and Dilithium signature verification:
    - i. Begins discovery operations on the VM
    - ii. Saves results to a structured JSON report file

## Results:

The demo produces a report (pqc\_secure\_discovery.json) containing:

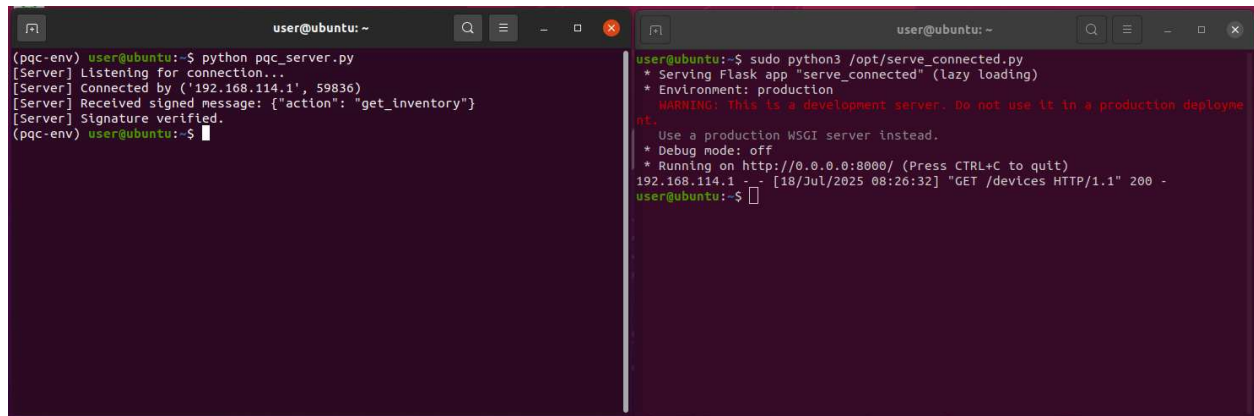
1. TLS scan status (if reachable)
  2. SSH banner from the target
  3. SNMP query result
  4. List of connected internal assets from the Flask API
- All actions are done only after a successful PQC handshake

### Note:

The demo can also be performed using just a regular computer. This can be done by changing the IP address in the scripts to local host and then setting up a terminal for the server and the other for the client and discovery agent.

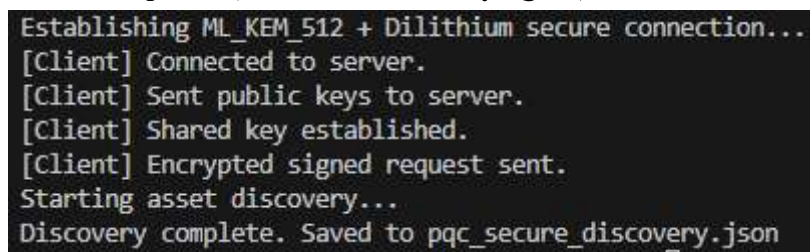
### Demo Visuals:

#### VM (Acts as the server)



The image shows two terminal windows side-by-side. The left window, titled 'user@ubuntu: ~', shows the execution of 'python pqc\_server.py'. It displays messages: '[Server] Listening for connection...', '[Server] Connected by ('192.168.114.1', 59836)', '[Server] Received signed message: {"action": "get\_inventory"}', and '[Server] Signature verified.' The right window, also titled 'user@ubuntu: ~', shows the execution of 'sudo python3 /opt/serve\_connected.py'. It displays messages: '\* Serving Flask app "serve\_connected" (lazy loading)', '\* Environment: production', a red warning 'WARNING: This is a development server. Do not use it in a production deployment.', and '\* Use a production WSGI server instead.' It also shows '\* Debug mode: off', '\* Running on http://0.0.0.0:8000/ (Press CTRL+C to quit)', and a log entry '192.168.114.1 - - [18/Jul/2025 08:26:32] "GET /devices HTTP/1.1" 200 -'.

#### Local computer (hosts the discovery agent)



The image shows a terminal window with the following output: 'Establishing ML\_KEM\_512 + Dilithium secure connection...', '[Client] Connected to server.', '[Client] Sent public keys to server.', '[Client] Shared key established.', '[Client] Encrypted signed request sent.', 'Starting asset discovery...', and 'Discovery complete. Saved to pqc\_secure\_discovery.json'.

### Result:

Results can be seen in the pqc\_secure\_discovery.json file.