

Post-Quantum

Cryptography Conference

Hybrid Quantum-Safe Cryptography for Electric Vehicle Charging Infrastructure



Alessandro Amadori

Cryptographer at TNO

KEYFACTOR

CRYPTO4A

SSL.com

ENTRUST

HID

October 28 - 30, 2025 - Kuala Lumpur, Malaysia

PKI Consortium Inc. is registered as a 501(c)(6) non-profit entity ("business league") under Utah law (10462204-0140) | pkic.org

 **PKI**
Consortium

Hybrid Quantum-Safe Cryptography for Electric Vehicle Charging Infrastructure

Dr. A. Amadori



[Start presentation](#)

Goal

- The DITM project
- The EnergyPod
 - Quest for PQC
- Chosen use-case: OCPP
 - Proof-of-concept
- Our migration approach
- Takeaways

What is the DITM project?

Digital Infrastructure for (T)future-proof Mobility

Intro Video: [/watch?v=eraeEnvv6o0](#)

- Research project led by Brainport Eindhoven, the Netherlands
 - made possible by the Ministry of Infrastructure and Water Management (I&W)
- 4 years and 60 million Euros
- 20 Partners:
 - Brainport Development, Heliox, Eindhoven University of Technology, ElaadNL, ELEO, Geomaat B.V., InQuisitive, Infiniot, Monotch | We Make Traffic Talk, Nationaal Kennisplatform Laadinfrastructuur NKL, NXP Semiconductors, RAI Automotive Industry NL, Scholt Energy, Siemens, Sioux Technologies, TNO, TomTom, VDL Groep, V-Tron B.V.
- Co-funded by European Union NextGenerationEU and DITM partners

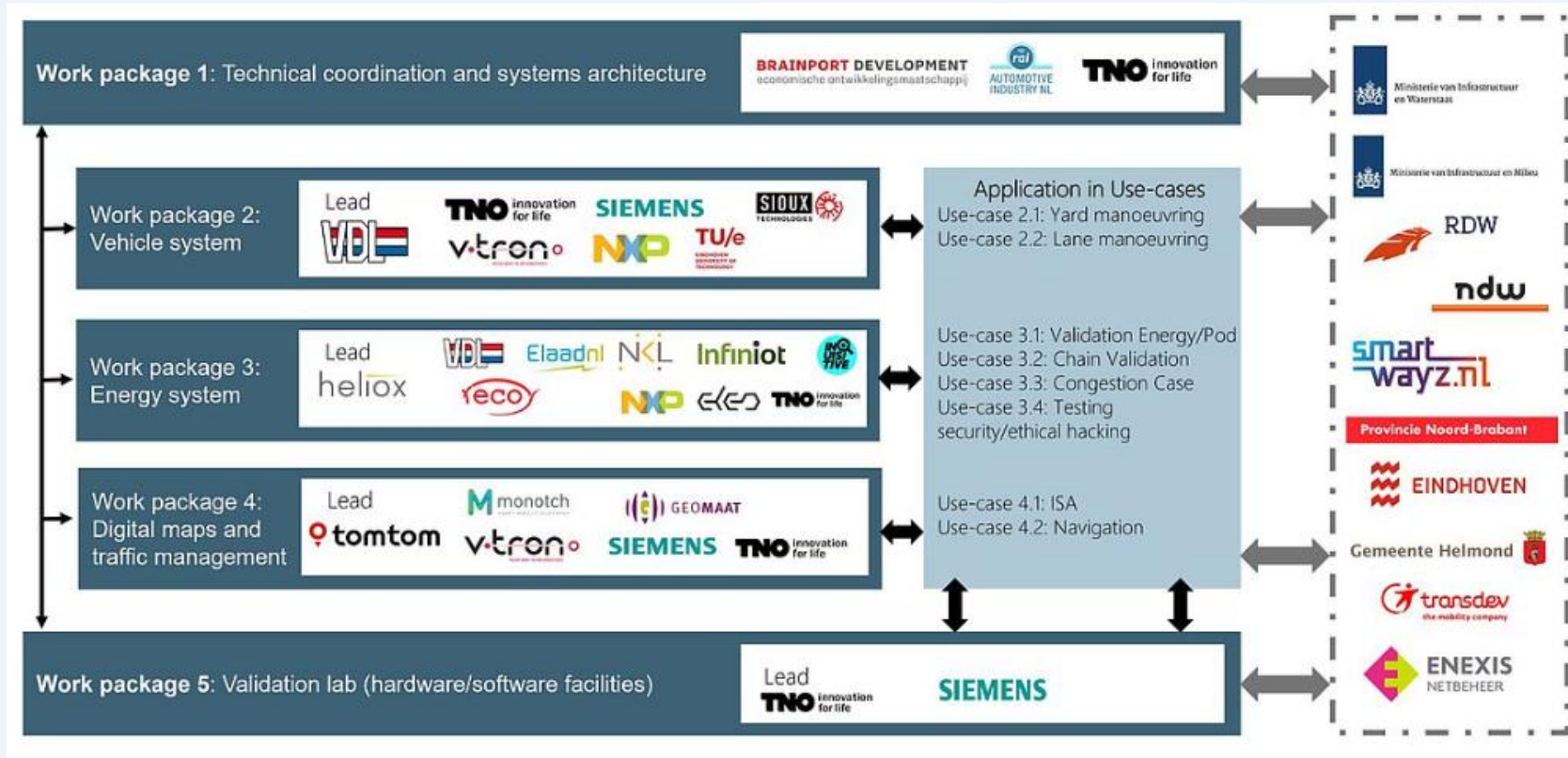
Goal of the project

The goal of the project is the creation of:

- **Autonomous functions related to connectivity, localization, and navigation of different vehicle platforms** (bus, passenger car) in different use cases (yard maneuvering, lane navigation, automatic approach and stopping at bus stops, and platooning, etc)
- An **“energy exchange system” (EnergyPod)** as a crucial link between autonomous electric vehicles (EVs) and the electricity grid, offering energy to grid operators in a cyber secure and reliable way.
- An innovative **map production system** that produces and updates maps based on continuous sensor observations of the road infrastructure and the increasing number of 'smart' vehicles on the road.
- A **Validation Lab** consisting of a virtual simulation and a physical test.

The structure

- Work-packages reflect the goals:
- Focus on security: WP3 Energy System Secure-by-design against present and future threats



Vision of the EnergyPod

- Future: 100 cities, ~50 buses per city; all charging at night. This requires orchestration, to preserve power grid safety.
 - Already plans for integration in Eindhoven
- Charging should happen at the cheapest tariff
- Power grid safety, integrity and availability should be guaranteed
- The local charging hub (DC-DC) and its back-office will communicate with bus station management offices down the stream and with energy providers and power grid management offices upstream.
- **All communication, from vehicle to grid-control needs to be cybersecure.**
- **Authentication should be compatible with existing EV standards.**

EnergyPod

The architecture

BESS: Battery Energy Storage System

BSP: Balancing Service Provider

CPO: Charging Point Operator

DSO: Distribution System Operator

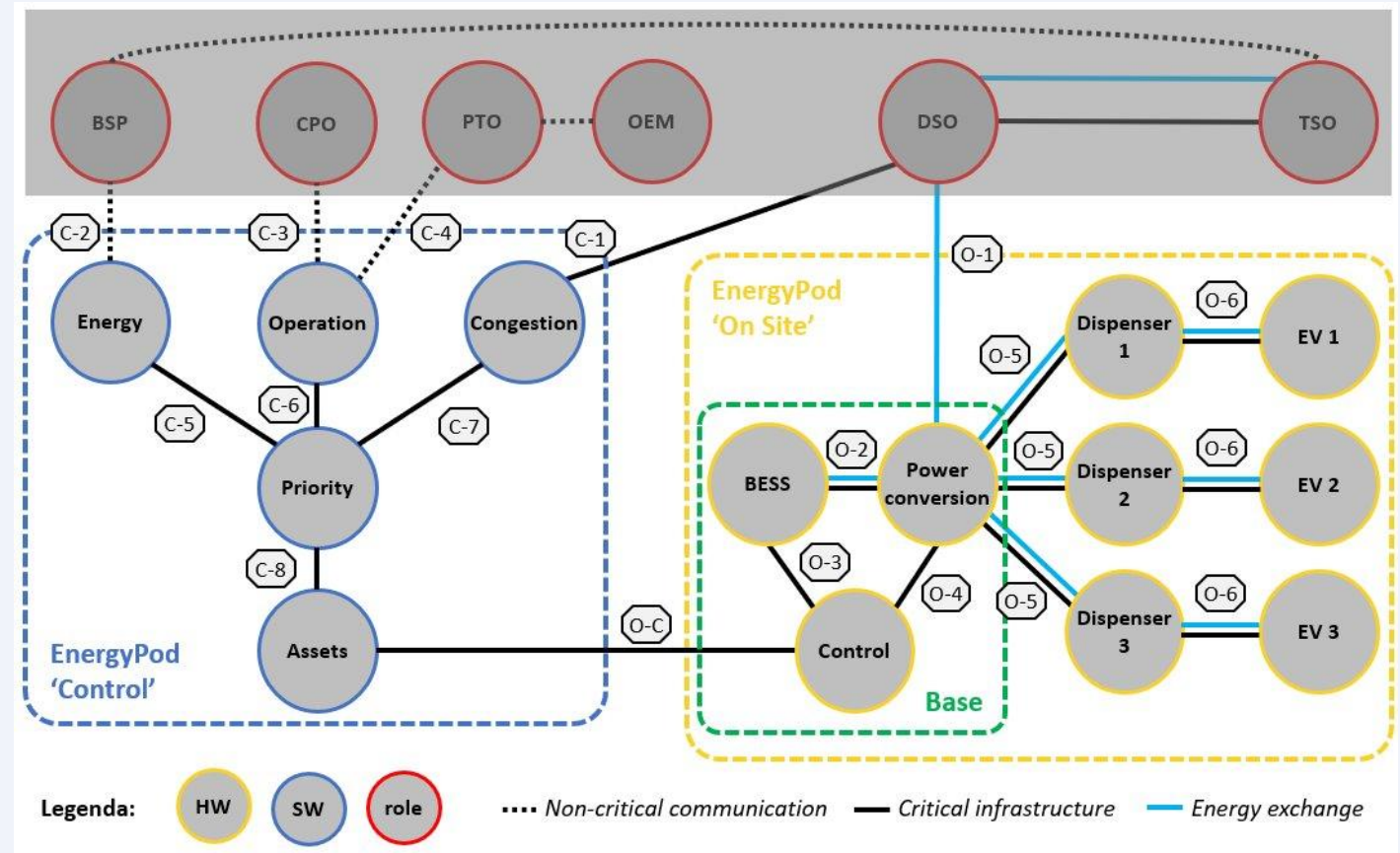
EV: Electric Vehicle

ESVE: Electric Vehicle Supply Equipment

OEM: Original Equipment Manufacturer

PTO: Public Transport Operator

TSO: Transport System Operator



- EnergyPod as a crucial link between the autonomous EVs and the environment (DSO/TSO).
- The EnergyPod makes both the physical and digital links that are essential to be able to distribute energy and information securely (cyber security) - according to the requirements of the grid operators – between EV-storage / BESS through the grid (DSO/TSO).

PQC Migration Handbook

When to migrate?

If we are dealing with:

- Sensitive Data
- Personal Data
- Critical infrastructures
- Long-lived infrastructures



As soon as possible!

The PQC Migration Handbook

GUIDELINES FOR MIGRATING TO POST-QUANTUM CRYPTOGRAPHY

Revised and Extended Second Edition

December, 2024

PQC Migration Handbook

When to migrate?

If we are dealing with:

- Sensitive Data
- Personal Data
- Critical infrastructures (Energy grid)
- Long-lived infrastructures (Charging stations)



As soon as possible!

EnergyPod must be secure-by-design against present and future threats

The PQC Migration Handbook

GUIDELINES FOR MIGRATING TO POST-QUANTUM CRYPTOGRAPHY

Revised and Extended Second Edition

December, 2024

European Commission Roadmap

When to migrate?

If we are dealing with:

- High risk

by 2030!

EnergyPod must be secure-by-design against present and future threats

Timeline for the transition to PQC

1. By **31.12.2026**:

- At least the *First Steps* have been implemented by all Member States.
- Initial national PQC transition roadmaps have been established by all Member States.
- PQC transition planning and pilots for high- and medium-risk use cases have been initiated.

2. By **31.12.2030**:

- The *Next Steps* have been implemented by all Member States.
- The PQC transition for high-risk use cases has been completed.
- PQC transition planning and pilots for medium-risk use cases have been completed.
- Quantum-safe software and firmware upgrades are enabled by default.

3. By **31.12.2035**:

- The PQC transition for medium-risk use cases has been completed.
- The PQC transition for low-risk use cases has been completed as much as feasible.

Open Charging Point Protocol

Open Charging Point Protocol

- [Wat is OCPP?](#) (ElaadNL)
- Open Standard maintained by the Open Charge Alliance
- Protocol for the communication between a Charging Station and back-office system (CSMS).
 - Status reporting
 - Charging process
 - Based on the omni-present cryptographic protocol TLS.



8.1. Charging Station(s) directly connected to CSMS

Description

This is the basic setup for using OCPP.

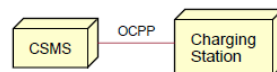
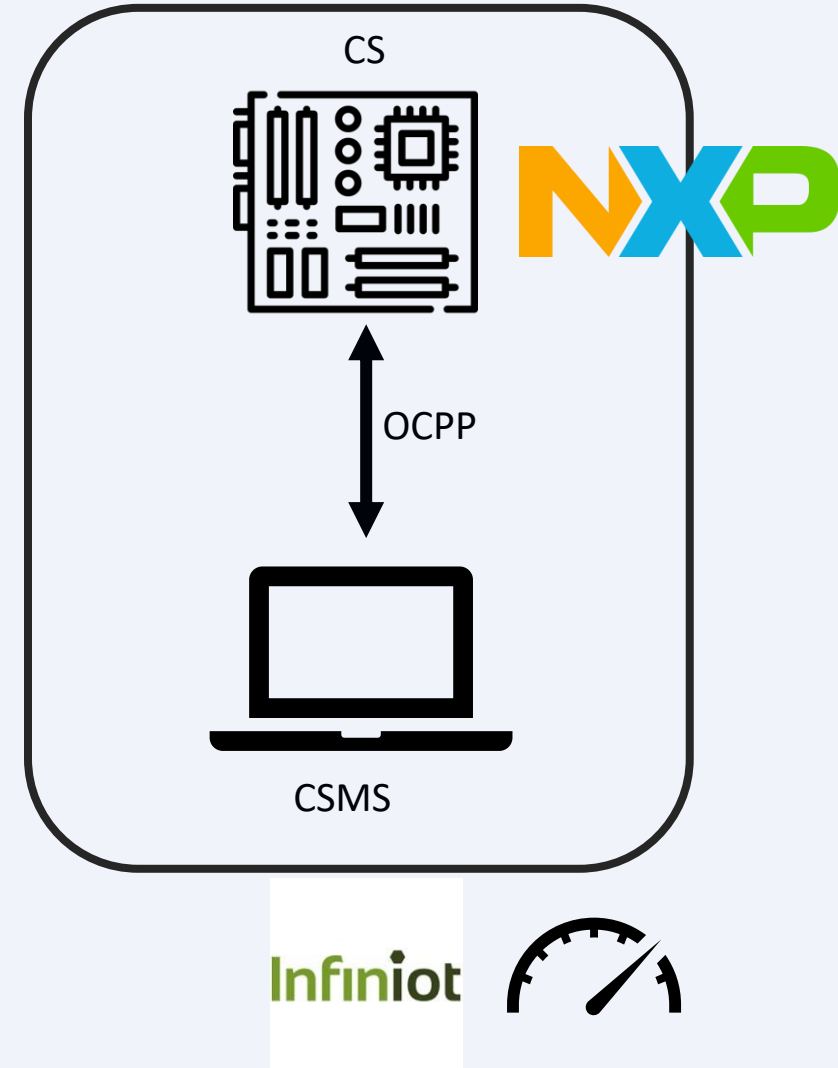


Figure 7. Charging Station directly connected to CSMS

Goal of the Proof-of-concept

Based on the [opensource python implementation by Mobility House](#)

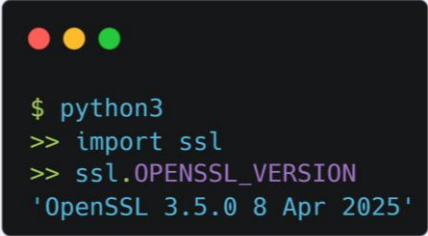
- PQC with a fundamental protocol running on deployable components
 - Enabling PQC inside of Python applications
 - Upgrading Openssl with OQS provider, linking it to Python SSL
- First modelled with Raspberry PI
 - Preparatory step for the NXP i.MX 8 board (NXP)
- Creation of a visualization dashboard (Infiniot)
 - Integration within existing validation infrastructure
 - Monitoring the use of PQC within the infrastructure
 - Creation of a visual way of representing PQC migration
- Testing several migration scenarios
 - Classical-PQC-hybrid



Use-case: Migrating OCPP

Demo: OCPP with hybrid Post-quantum Cryptography

- Dockerized Python implementation
 - Python's ssl linked to OpenSSL 3.5
- Hybrid PQC implementations
 - Hybrid KEM for secure connection
 - Hybrid PKI for authentication (with use of oqs-provider)
- Tested on NXP i.MX 8 boards



```
$ python3
>> import ssl
>> ssl.OPENSSL_VERSION
'OpenSSL 3.5.0 8 Apr 2025'
```

Use-case: Migrating OCPP

Demo: OCPP with hybrid Post-quantum Cryptography

- Dockerized Python implementation
 - Python's ssl linked to OpenSSL 3.5
- Hybrid PQC implementations
 - Hybrid KEM for secure connection
 - Hybrid PKI for authentication (with use of oqs-provider)
- Tested on NXP i.MX 8 boards

```
$ openssl list -providers
```

```
Providers:
```

```
  default
```

```
    name: OpenSSL Default Provider
```

```
    version: 3.5.0
```

```
    status: active
```

```
  oqsprovider
```

```
    name: OpenSSL OQS Provider
```

```
    version: 0.10.1-dev
```

```
    status: active
```

Use-case: Migrating OCPP

Demo: OCPP with hybrid Post-quantum Cryptography

- Dockerized Python implementation
 - Python's ssl linked to OpenSSL 3.5
- Hybrid PQC implementations
 - Hybrid KEM for secure connection
 - Hybrid PKI for authentication (with use of oqs-provider)
- Tested on NXP i.MX 8 boards

```
$ openssl list -kem-algorithms

{ 1.2.840.113549.1.1.1, 2.5.8.1.1, RSA, rsaEncryption } @ default
{ 1.2.840.10045.2.1, EC, id-ecPublicKey } @ default
{ 1.3.101.110, X25519 } @ default
{ 1.3.101.111, X448 } @ default
{ 2.16.840.1.101.3.4.4.1, id-alg-ml-kem-512, ML-KEM-512, MLKEM512 } @ default
{ 2.16.840.1.101.3.4.4.2, id-alg-ml-kem-768, ML-KEM-768, MLKEM768 } @ default
{ 2.16.840.1.101.3.4.4.3, id-alg-ml-kem-1024, ML-KEM-1024, MLKEM1024 } @ default
X25519MLKEM768 @ default
X448MLKEM1024 @ default
SecP256r1MLKEM768 @ default
SecP384r1MLKEM1024 @ default
```


Use-case: Migrating OCPP

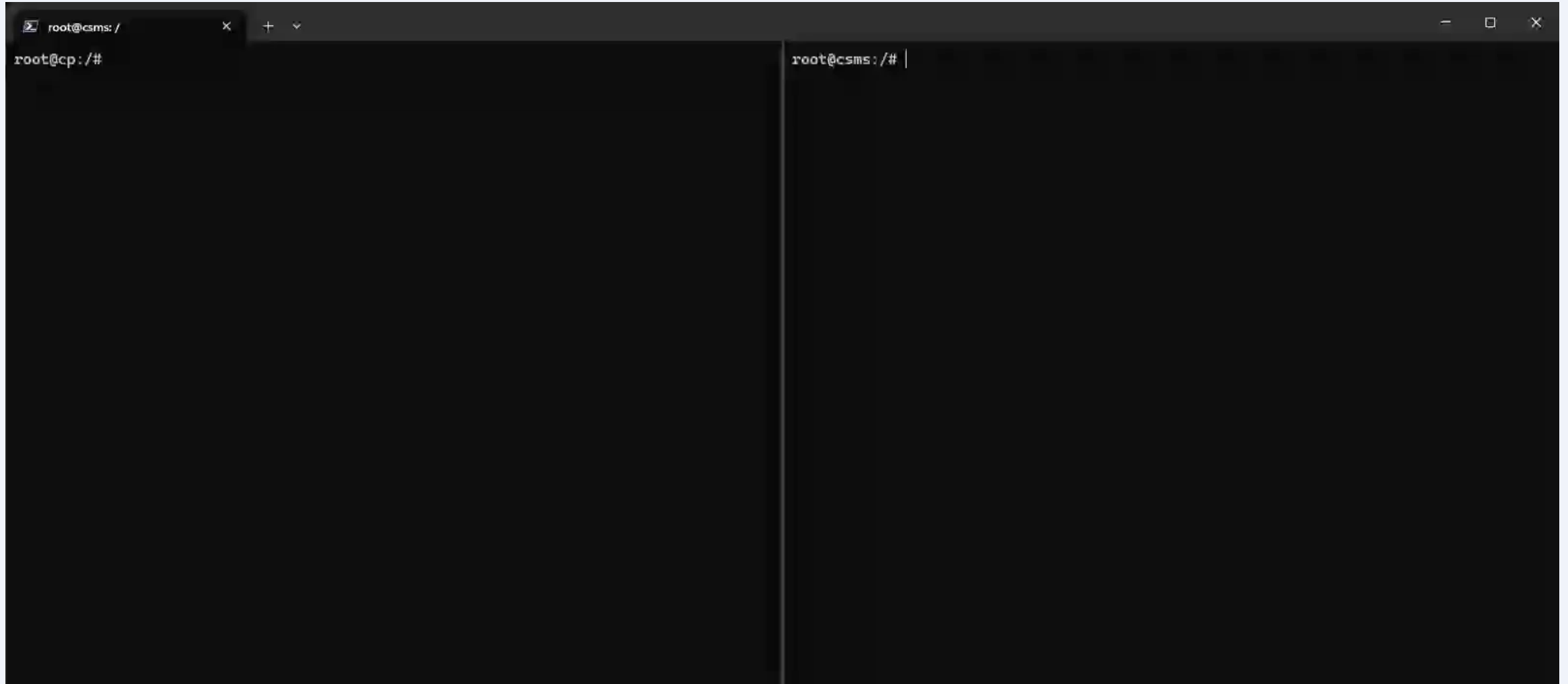
Demo: OCPP with hybrid Post-quantum Cryptography

- Dockerized Python implementation
 - Python's ssl linked to OpenSSL 3.5
- Hybrid PQC implementations
 - Hybrid KEM for secure connection
 - Hybrid PKI for authentication (with use of oqs-provider)
- Tested on NXP i.MX 8 boards

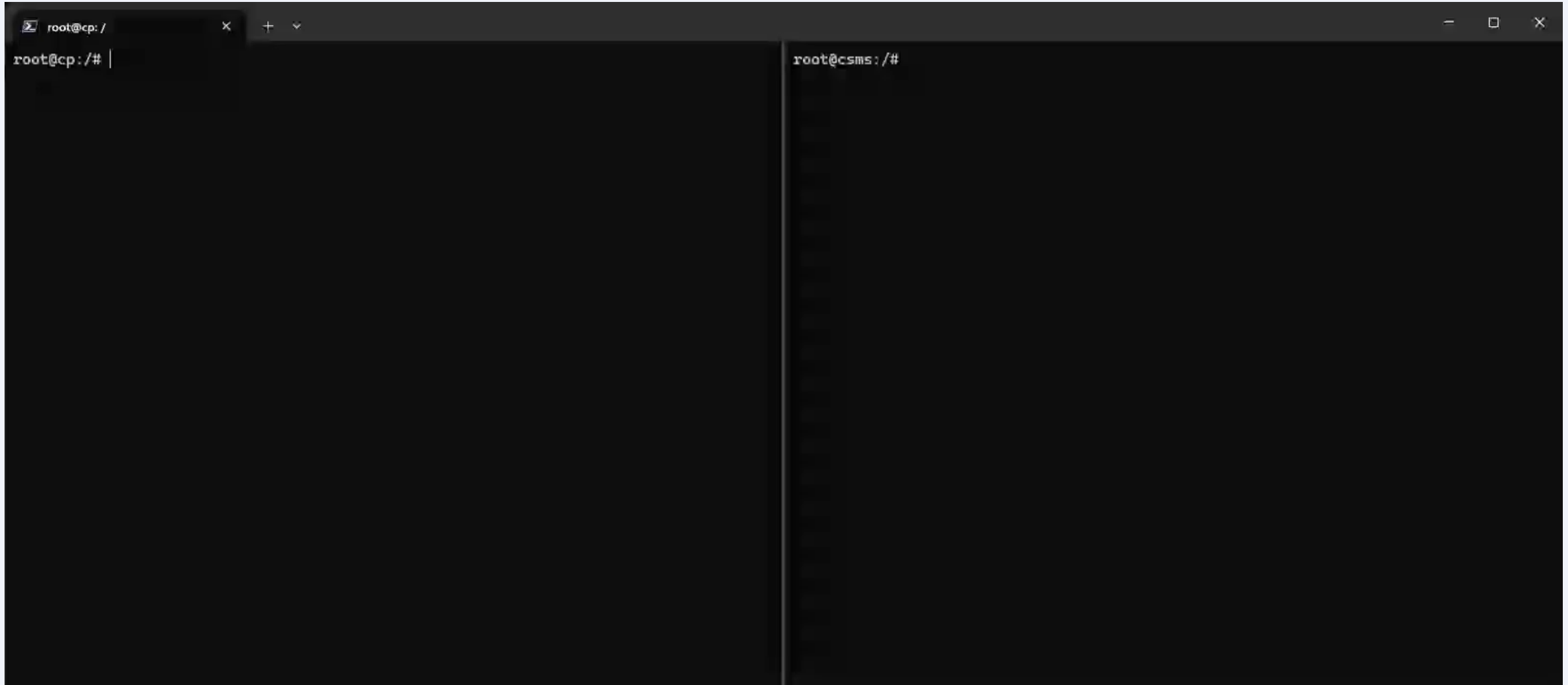
```
$openssl list -public-key-algorithms | grep mldsa
```

```
Name: p256_mldsa44  
IDs: p256_mldsa44 @ oqsprovider  
Name: rsa3072_mldsa44  
IDs: rsa3072_mldsa44 @ oqsprovider  
Name: p384_mldsa65  
IDs: p384_mldsa65 @ oqsprovider  
Name: p521_mldsa87  
IDs: p521_mldsa87 @ oqsprovider
```

Hybrid PQC communication



Failing communication



Logging code (python)

- TLS 1.3 encrypts the certificate exchange
 - Cannot identify authentication algorithms
- Logging is used to record public information
 - Certificate signatures and public keys
- Information digested by the dashboard

```
# Previously agreed upon certificate for dashboard
ssl_context = ssl.create_default_context(cafile="digicert_root.pem")

# Setting up syslog HTTPHandler
http_handler = HTTPHandler(host=os.environ['HTTP_HANDLER_HOST'], url=os.environ['HTTP_HANDLER_URL'],
                           method='POST', secure=True, context=ssl_context)

async with websockets.connect(
    "wss://csms:9000/CP_1", subprotocols=["ocpp2.0.1"], ssl=ssl_context
) as wss:
    # Get ssl_object out of websocket
    ssl_obj = wss.transport.get_extra_info('ssl_object')

    # Get peer certificate that we have authenticated and print information through CLI
    cert_der = ssl_obj.getpeercert(binary_form=True)
    cert = subprocess.run(['openssl', 'x509', '-noout', '-text'],
                          input=cert_der, stdout=subprocess.PIPE)

    # Search algorithm data
    cert_str = cert.stdout.decode('utf-8')
    re_pkey = re.search("Public Key Algorithm: .*", cert_str)
    re_signalgo = re.search("Signature Algorithm: .*", cert_str)

    # If found, log algorithms used
    if re_pkey:
        httplogger.info(re_pkey.group())

    if re_signalgo:
        httplogger.info(re_signalgo.group())
```


Logging output

Not complete or realistic but a selection of logs

- TLS is being used thanks to the ssl-contexts, but it is not logged within python.
- We don't know which KEM is used
- Need to rely on Packet Sniffers
- We can retrieve the digital certificate and read the algorithms included

```

csms-1      | 2025-09-18 12:41:04,302      websockets.server      INFO      'server listening on 172.19.0.3:9000'
csms-1      | 2025-09-18 12:41:04,302      root                  INFO      'Server Started listening to new connections...'

cp-1        | 2025-09-18 12:41:09,281      websockets.client      DEBUG     '= connection is CONNECTING'
cp-1        | 2025-09-18 12:41:09,302      websockets.client      DEBUG     '> GET /CP_1 HTTP/1.1'
cp-1        | 2025-09-18 12:41:09,302      websockets.client      DEBUG     '> Upgrade: websocket'
cp-1        | 2025-09-18 12:41:09,302      websockets.client      DEBUG     '> Connection: Upgrade'

csms-1      | 2025-09-18 12:41:09,310      websockets.server      DEBUG     '< Connection: Upgrade'
csms-1      | 2025-09-18 12:41:09,310      websockets.server      DEBUG     '< Sec-WebSocket-Version: 13'
csms-1      | 2025-09-18 12:41:09,311      websockets.server      DEBUG     '< Sec-WebSocket-Protocol: ocpp2.0.1'
csms-1      | 2025-09-18 12:41:09,311      websockets.server      DEBUG     '< User-Agent: Python/3.12 websockets/15.0.1'
csms-1      | 2025-09-18 12:41:09,312      websockets.server      DEBUG     '> HTTP/1.1 101 Switching Protocols'
csms-1      | 2025-09-18 12:41:09,312      websockets.server      DEBUG     '> Upgrade: websocket'
csms-1      | 2025-09-18 12:41:09,312      websockets.server      DEBUG     '> Connection: Upgrade'
csms-1      | 2025-09-18 12:41:09,313      websockets.server      DEBUG     '= connection is OPEN'

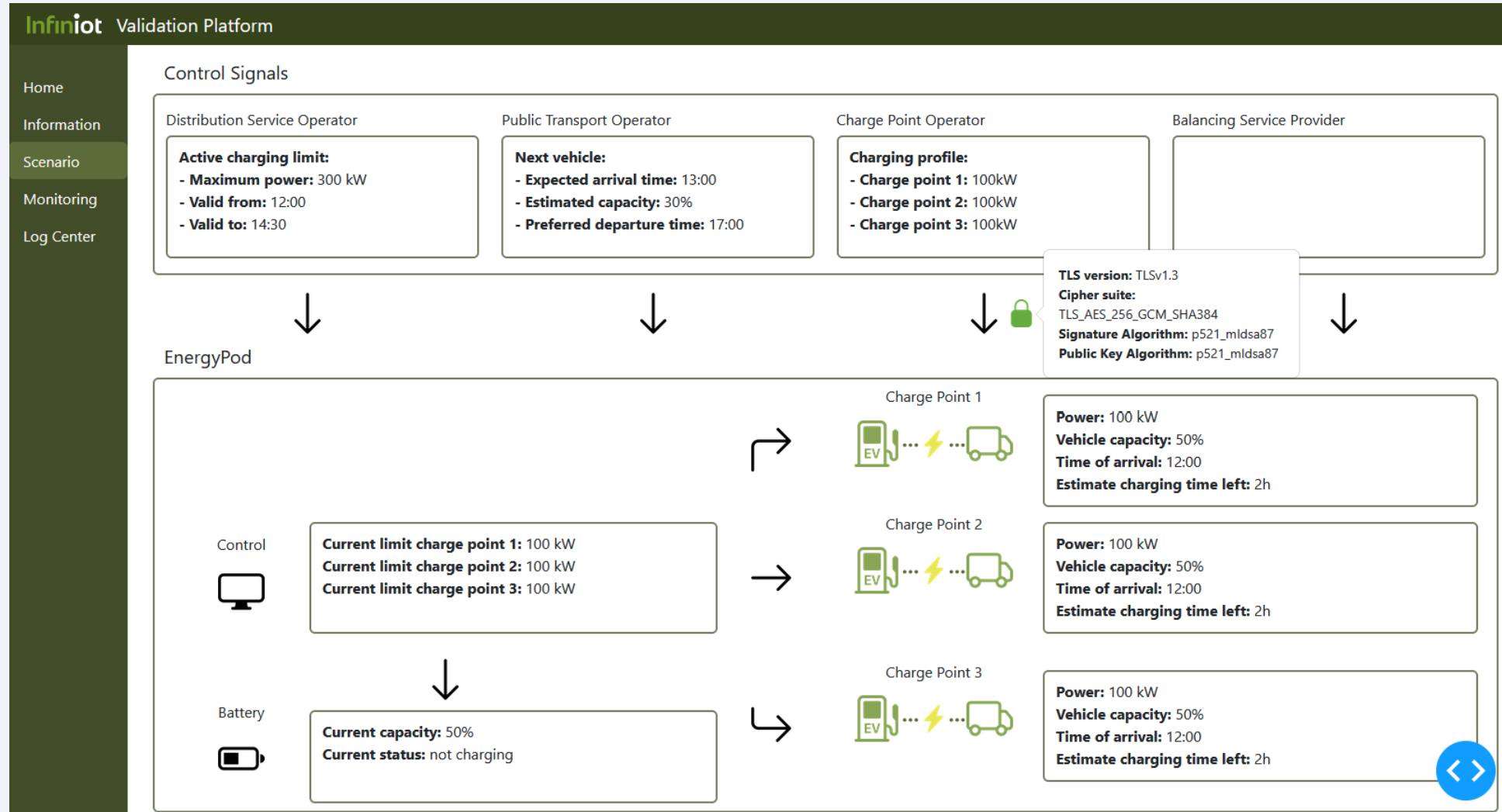
cp-1        | 2025-09-18 12:41:09,315      root                  INFO      'TLSv1.3'
cp-1        | 2025-09-18 12:41:09,359      root                  INFO      'Public Key Algorithm: p521_mldsa87'
cp-1        | 2025-09-18 12:41:09,359      root                  INFO      'Signature Algorithm: p521_mldsa87'

(...)

cp-1        | 2025-09-18 12:41:09,362      ocpp                  INFO      'CP_1: send [2,"33c4b1dc-cf6c-4c74-9832-
e5c143b3f61a","BootNotification",{
cp-1        | 2025-09-18 12:41:09,362      ocpp                  INFO      'CP_1: send [2,"33c4b1dc-cf6c-4c74-9832-
e5c143b3f61a","BootNotification",{

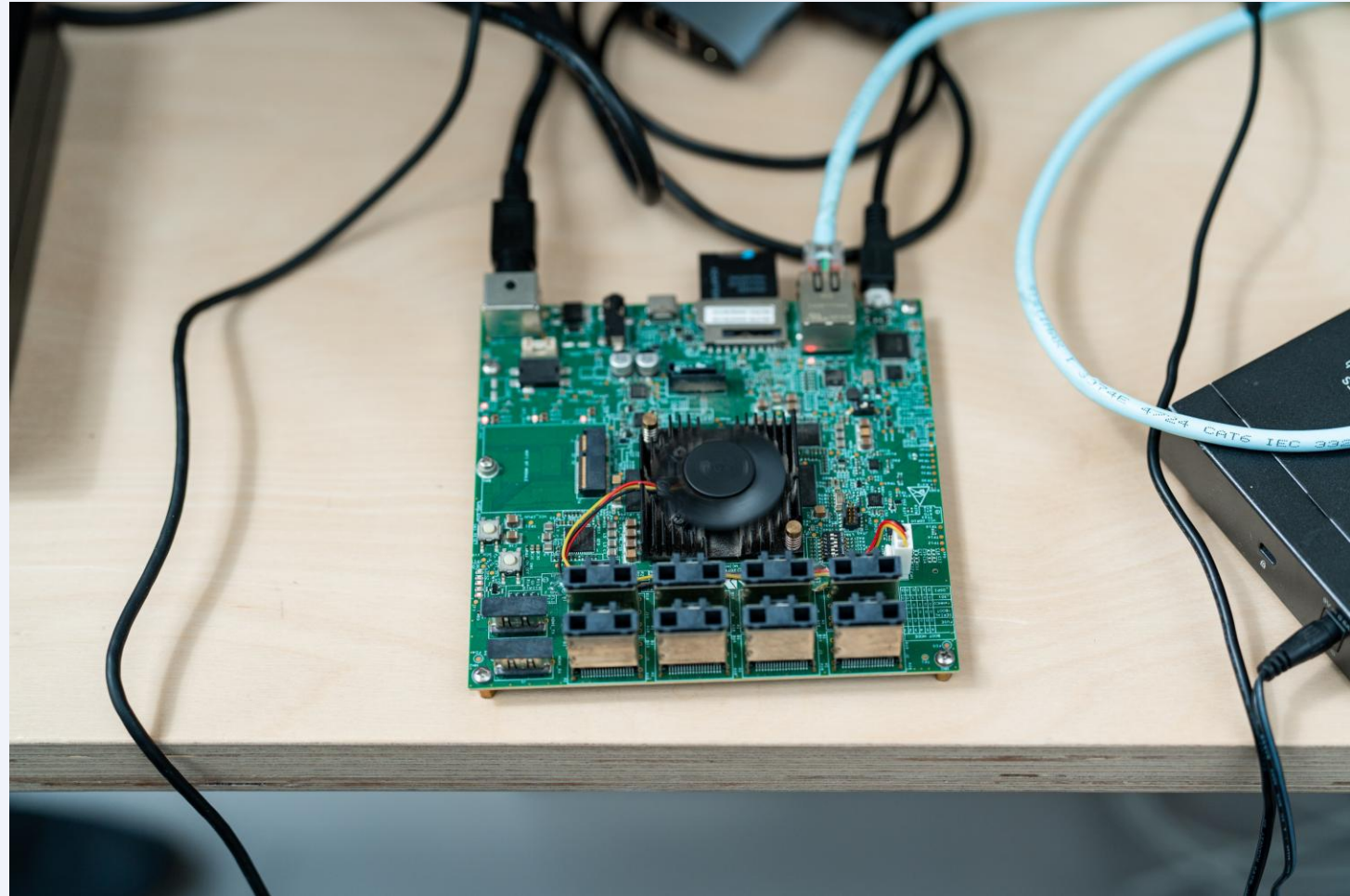
```

The Dashboard



Demo in action

- i.MX 8 NXP board



Demo in Action

- Successful PQC connection
- CP and CSMS



Demo in Action

- Real-time connection monitoring
- Via Infiniot's dashboard



Takeaways

- Migration was simple by making Python use OpenSSL 3.5, and the ability of the sockets library to use ssl-contexts.
 - Migrate to OpenSSL 3.5 ASAP!
- The good news is that migrating Python applications to PQC is (fairly) straightforward
 - Bottleneck: lack of hybrid signature standards
- Logging the use of cryptography is a good first step into monitoring the transition of your infrastructure and creating an inventory of cryptographic assets
 - Negotiation not so easy to log through Python
 - Need for packet sniffers or OpenSSL's CLI
 - New best practices for cryptographic management?



Thank you