Multi-Party Domain Cryptosystem

MPDC-I Technology Integration Guide

Revision: 1.0
Date: October 14, 2025

1 Introduction

The **Multi-Party Domain Cryptosystem – Interior protocol (MPDC-I)** is a post-quantum, multi-party key exchange and network security system. Instead of relying on a single key-exchange between a client and server, MPDC-I distributes the exchange across several independently authenticated devices: **root domain security (RDS)**, **domain list agent (DLA)**, **managed application server (MAS)**, **Agents**, and **Clients**. Each agent contributes pseudo-random fragments that are combined by the MAS and client using a hybrid post-quantum scheme (Kyber/McEliece KEMs, Dilithium/SPHINCS+ signatures) and hashed via cSHAKE to derive session keys. This distributed entropy injection means an adversary would need to compromise multiple devices at once to reconstruct the session key, making impersonation and man-in-the-middle attacks extremely difficult. MPDC-I employs a symmetric cipher (RCS) with cSHAKE-based key expansion and KMAC authentication to encrypt and authenticate traffic. The design is scalable, computationally efficient and resistant to both classical and quantum attacks.

This guide explains MPDC-I's architecture, describes the API and main functions, and provides practical steps for deploying MPDC in payment networks, cloud platforms, SCADA/industrial systems and IoT environments.

2 Architecture and Roles

MPDC-I operates with five device types:

Entity	*Role*	*Key responsibilities*
Client	End-user device initiating secure communication	Generates a certificate, gets it signed by RDS (directly or via DLA), exchanges master fragment keys (mfk) with agents and MAS, combines key fragments to derive session keys and encrypts/decrypts traffic.

Managed Application Server (MAS)	Central application server	Generates its own certificate, validates client certificates, contacts agents to obtain key fragments, combines them with its MAS fragment key to derive session keys and operates the encrypted tunnel.
Agent	Trusted network device injecting entropy	Generates its own certificate, obtains a root-signed certificate, produces key fragments using symmetric ciphers, and transmits fragments to MAS and clients. Agents may run in separate physical devices to diversify entropy sources.
Domain List Agent (DLA)	Network registry	Holds the network topology and certificate list, verifies device certificates, issues incremental updates, manages join/resign requests and revocation broadcasts.
Root Domain Security (RDS)	Root certificate authority	Generates and manages the root certificate, signs device certificates, and acts as the ultimate trust anchor. RDS should operate in a secure environment and may act via DLA proxies for convenience.

Key exchange and session derivation

During an MPDC-I session, the client and MAS independently perform asymmetric KEM exchanges with each agent to obtain per-agent shared secrets. Each agent uses its symmetric key stream (derived from the KEM secret) to generate **key fragments**. The client and MAS collect all fragments, combine them along with a MAS-specific fragment key, and feed them into cSHAKE to derive the final session keys. Because every session key depends on contributions from multiple agents, compromising a single device does not expose the entire session key. RCS with KMAC authenticates and encrypts traffic, while Dilithium/SPHINCS+ signatures authenticate the initial certificate and KEM keys. The hybrid design provides strong forward secrecy and quantum resistance.

Topology management

The DLA maintains a master list of devices and distributes certificates, updates and revocations. Each device holds a **child certificate** signed by the RDS, indicating its role and expiration. The MAS and clients validate incoming certificates using the root certificate. **Port numbers** for each component are defined in mpdc.h: clients default to port 37761, DLA 37762, IDG 37763, RDS 37764 and MAS 37765.

3 Cryptographic and Certificate APIs

MPDC-I maps directly to QSC's cryptographic primitives via macros in mpdc.h. Key functions include:

Certificate creation and management

MPDC's certificate API allows creation, serialization, encoding and verification of root and child certificates. Important functions are:

- **mpdc_certificate_root_create(root, pubkey, expiration, issuer):** create a root certificate structure with the given public key, expiration and issuer.
- **mpdc_certificate_root_encode(enck, root) / mpdc_certificate_root_decode(root, enck):** convert between a root certificate structure and its human-readable string representation.
- **mpdc_certificate_child_create(child, pubkey, expiration, issuer, designation):** create a child certificate specifying the device designation (client, MAS, agent or DLA).
- **mpdc_certificate_child_encode(enck, child) / mpdc_certificate_child_decode(child, enck):** encode or decode child certificates.
- **mpdc_certificate_child_is_valid(child):** verify format and expiration.
- **mpdc_certificate_message_hash_sign(signature, sigkey, message, msglen)** and **mpdc_certificate_child_message_verify(message, msglen, signature, siglen, child):** sign and verify messages using certificate keys.

4 Server lifecycle functions

Each MPDC role provides start, pause and stop functions. These functions initialize network resources, load certificates, start command loops and shut down gracefully. Key functions include:

Role	Start	Pause	Stop	Description
Client	mpdc_client_start_server()	mpdc_client_pause_server()	mpdc_client_stop_server()	Initializes sockets, loads client certificate, registers with the network and begins listening for connections. Use the callback

				mpdc_client_connect_callback() to accept or reject incoming connections.
MAS	mpdc_mas_start_server()	mpdc_mas_pause_server()	mpdc_mas_stop_server()	Loads configuration and certificates, initializes secure sockets and starts the command loop.
Agent	mpdc_agent_start_server()	mpdc_agent_pause_server()	mpdc_agent_stop_server()	Handles certificate management, topology convergence, fragment queries and master fragment key exchanges.
DLA	mpdc_dla_start_server()	mpdc_dla_pause_server()	mpdc_dla_stop_server()	Configures network listening sockets, loads DLA certificate, initializes topology database and command loop.
RDS	mpdc_rds_start_server()	mpdc_rds_pause_server()	mpdc_rds_stop_server()	Manages creation, storage and signing of root certificates and coordinates secure communications.

5 Example Configuration and Deployment Workflow

The Doxygen documentation provides a typical sequence for initializing an MPDC-I network. Key steps are summarized below. Note that configuration commands (enable, config, certificate, etc.) are executed via the console command loop provided by each server.

1. **RDS Initialization:**
    - Log into the RDS console and enter *enable* mode.
    - Configure user credentials, device name, IP address and network name.
    - Enter *configuration* then *certificate* mode. Generate the root certificate using generate(days-valid) specifying the validity period.
2. **DLA Initialization**:

- Log into the DLA console and enter *enable* mode.
- Provide the path to the root certificate generated by the RDS.
- Enter *certificate* mode and generate the DLA certificate.

3. **Sign the DLA Certificate**:
    - On the RDS, import the DLA certificate and sign it using the sign(certificate-path) command. Return the signed certificate to the DLA.
4. **Device Initialization (Agent, MAS, Client)**:
    - For each device, generate a keypair and certificate via mpdc_certificate_child_create(). Have the certificate signed by the RDS or via the DLA.
    - Register with the DLA (register(dla-ip-address)) to join the network and update the topology.
5. **MAS and Agent Integration**:
    - On the MAS, start the server (mpdc_mas_start_server()), then join the network by contacting the DLA and obtaining a list of Agents.
    - On each Agent, start the server and register with the DLA using register(dla-ip-address).
6. **Client Integration**:
    - On each client, enable the network service and register with the DLA.
    - Exchange certificates and master fragment keys (mfk) with Agents and the MAS. The mpdc_client_connect_callback() can be used to evaluate incoming connections.
    - To establish a secure session, run connect(mas-ip-address) in server mode. The MAS will validate the client certificate and complete the key exchange.

After initialization, clients and MAS derive session keys by combining key fragments from all participating agents and the MAS's own fragment key. Each secure tunnel uses RCS with KMAC to provide authenticated encryption and includes sequence numbers and timestamps to prevent replay attacks.

6 Integration Scenarios

Payment Networks

MPDC-I can replace or augment existing payment network security infrastructures. By distributing key derivation across multiple agents, the protocol reduces reliance on central HSMs and mitigates breach scope. MAS servers can be deployed at payment processors, clients in point-of-sale (POS) terminals, and agents in ATM clusters. The DLA maintains the device registry and certificate revocation list. Each transaction uses new session keys derived from fresh agent fragments, ensuring forward secrecy and post-quantum protection. Integration steps include:

1. Deploy an RDS within the payment operator's secure facility and establish DLA servers to manage POS and ATM certificates.
2. Issue root-signed certificates to MAS (processor), agents (regional HSMs), and clients (POS/ATM terminals).
3. Use the mpdc_client_start_server() API in the POS firmware to initialize the MPDC client and automatically register with the DLA during provisioning.
4. For each payment session, the client requests fresh fragments from agents and the MAS, combines them and derives session keys; transactions are encrypted using RCS with KMAC.

Cloud Platforms and SaaS

In cloud environments, MPDC-I provides a sovereignty-preserving trust fabric across micro-services. MAS servers run in each data-center, while agents may be deployed as separate service nodes (or container side-cars) generating entropy. Clients can be API gateways or service consumers. The DLA maintains a registry of services and orchestrates revocations. The low overhead of MPDC's key exchange (symmetric operations for fragments) allows large scale adoption without significant CPU impact. Integration involves embedding the MPDC client library into service frameworks and using MAS as secure ingress points. Certificates and trust structures can be stored in secrets managers, with rotation handled via the command loop.

SCADA / Industrial Control

MPDC-I is well suited for critical infrastructure because it distributes trust and does not rely on always-online certificate authorities. Agents can be placed at substations or control units to inject locally generated randomness. MAS servers coordinate control commands and monitoring. Clients reside on operator consoles or remote sensors. In this context, the RDS may be air-gapped and DLA updates are pushed during maintenance windows. The mpdc_agent_start_server() and mpdc_agent_stop_server() APIs allow operators to register or

decommission agents safely. Sequence numbers and timestamps in each RCS-encrypted packet prevent replay and ensure command integrity.

IoT Networks

For IoT devices requiring long-term security and minimal overhead, MPDC-I can be combined with lightweight QSMP or SKDP transport. The client module can be compiled with MISRA C to run on embedded controllers. Agents may be small hubs or gateways providing additional entropy. MAS servers run in central aggregators or cloud services. Because MPDC uses symmetric cryptography for most operations, low-power devices can participate in multi-party key exchange. Device provisioning includes generating a child certificate via mpdc_certificate_child_create() and registering with the DLA; subsequent communication uses the same session derivation as other domains.

7 Security and Best Practices

- **Distributed entropy:** Always deploy multiple agents; each session key should derive from at least one MAS fragment and multiple agent fragments to ensure no single compromise exposes the key.
- **Certificate management:** Keep the RDS offline except during certificate signing operations. Use mpdc_certificate_expiration_set_days() or mpdc_certificate_expiration_set_seconds() to set appropriate validity periods. Regularly rotate child certificates and update the DLA topology.
- **Port isolation:** Use default port assignments defined in mpdc.h or configure custom ports behind firewalls. Ensure each server only listens on its designated port to avoid cross-role confusion.
- **Command loop security:** The MPDC command loop supports different modes (user/config/certificate). Restrict access to the *enable* and *certificate* modes to privileged administrators. Use console idleness timeouts to log out inactive sessions.
- **Trust record management:** Use mpdc_trust_serialize() and mpdc_trust_deserialize() to back up and restore trust databases. Clear outdated entries with mpdc_trust_clear().
- **Hybrid cryptography:** MPDC-I uses Kyber for KEM and Dilithium for signatures by default via macros in mpdc.h. Monitor NIST PQC standard updates and update algorithm sets as needed. The certificate API (mpdc_certificate_algorithm_decode() / encode()) allows switching to alternative parameter sets.

- **Replay prevention:** Ensure that sequence numbers and timestamps included in each RCS packet are validated and that out-of-order or duplicate packets are rejected.

8 Conclusion

MPDC-I provides a resilient, scalable and quantum-secure alternative to traditional client-server key exchange. By distributing trust across multiple independently managed devices and combining asymmetric KEMs, hash-based signatures and symmetric ciphers, it resists quantum and classical adversaries. The well-structured API; ranging from certificate creation (mpdc_certificate_*), network registration (mpdc_dla_*), server life-cycle management (mpdc_*_start_server() etc.) and console help (mpdc_help_print_*), allows implementers to integrate MPDC-I into payment systems, cloud services, SCADA networks and IoT devices with relative ease. Following the configuration workflow and adhering to best practices ensures successful deployment and long-term security for sovereign infrastructures.