QRCS Corporation

Multi Party Domain Cryptosystem (MPDC) Analysis

Title: Implementation Analysis of the Multi Party Domain Cryptosystem (MPDC)
Author: John G. Underhill
Institution: Quantum Resistant Cryptographic Solutions Corporation (QRCS)
Date: November 2025
Document Type: Technical Cryptanalysis Report
Revision: 1.0

Chapter 1: Introduction

1.1 Identification of Artifacts

This cryptanalysis examines the interior protocol of the Multi-Party Domain Cryptosystem, designated MPDC-I, as defined in the authoritative specification *MPDC-2024 Rev.1b*. The specification states that MPDC-I is a multi-party authenticated key-exchange and network-security system intended for interior network environments. Its stated function is to establish an encrypted and authenticated duplex communication stream between a server, referred to as the MAS, and a client. The MAS and client do not perform a single asymmetric key exchange with each other; instead, they each maintain authenticated asymmetric relationships with multiple Agents, and these Agents contribute independent pseudo-random fragments that become inputs to the tunnel key derivation.

The analysis presented here is based on two primary artifacts:

1. The *MPDC-I Rev.1b* specification, which defines the protocol flows, certificate hierarchy, topology propagation, fragment exchange, key derivation, replay protection rules, message structures, and the expected behavior of each device class.

2. The accompanying C implementation, including *mpdc.h*, *mpdc.c*, *network.c*, *certificate.c*, *mas.c*, *client.c*, *agent.c*, *dla.c*, *rds.c*, and the symmetric and PQC routines in the QSC library. These files collectively implement the mechanisms described in the specification. All cryptographic operations, packet formats, and

message-handling rules appearing in the analysis are verified directly from these sources.

Throughout this document, the specification is treated as the normative description of intended protocol behavior, and the implementation is examined to confirm whether the behavior conforms to the specification without deviation. The analysis therefore assesses both correctness of design and correctness of implementation.

1.2 Protocol Purpose and High-Level Design Claims

The specification defines MPDC-I as a cryptosystem that distributes the security of a key exchange across multiple autonomous devices in a domain. Servers and clients share long-term authenticated secrets with Agents using post-quantum KEMs. Each Agent uses these long-term secrets to generate short pseudo-random fragments, encrypts and authenticates these fragments under symmetric derivations based on the master fragment keys, and transmits them to the MAS during a fragment-collection round.

The MAS aggregates the received fragments, verifies their authenticity, decrypts the MAS-side fragment for each Agent, and combines the contributions with its own fragment using a Keccak-based hash construction. The MAS then sends to the client a bundle containing the client-side ciphertexts and the MAS-derived fragment. The client repeats the fragment-verification, decryption, and aggregation steps. Both sides obtain an identical 256-bit value that becomes the seed for the two tunnel keys used in the subsequent encrypted channel.

The specification asserts that:

1. Session-key entropy is distributed among multiple authenticated network devices rather than created by a single handshake.
2. Fragment generation, fragment encryption, and aggregation use symmetric Keccak-based constructions to ensure scalability.
3. Certificates and control messages use post-quantum signature schemes and strict time and sequence controls to prevent replay or impersonation.
4. An adversary would need to compromise several authenticated devices in order to reconstruct the inputs to the tunnel-key derivation, which mitigates man-in-the-middle attacks and impersonation attacks.

These design claims form the basis for the security objectives evaluated in later chapters. Every claim is substantiated directly against the specification and the implementation.

1.3 Mechanism Surfaces to be Evaluated

The cryptanalysis evaluates MPDC-I across all surfaces defined in the specification and exercised by the implementation.

The certificate and topology subsystem defines a hierarchy anchored at the Root Domain Server. Devices hold signed child certificates containing the public verification key, issuer field, serial number, expiration interval, cryptographic configuration identifier, and version. The Domain List Agent distributes topology updates, certificate lists, revocations, convergence broadcasts, and node-status information. All such messages contain Dilithium signatures over a Keccak hash that includes the sequence number and timestamp of the packet. The implementation reconstructs these signed values and enforces all signature and time validations before modifying topology state.

The master fragment key subsystem uses authenticated post-quantum KEM exchanges between MAS and Agent, and between client and Agent. These exchanges bind signatures to the per-packet sub-header containing the sequence number and timestamp. The implementation verifies signatures, timestamps, certificate validity, and sub-header correctness before accepting any MFK material.

The fragment subsystem generates symmetric fragment-encryption keys using SHAKE-based derivations that incorporate the long-term MFK and round-specific values. Each fragment is masked under such a derived key and authenticated using KMAC. The MAS and client verify these values, decrypt them, and aggregate them into a hash that produces the tunnel-key seed. The implementation performs strict per-fragment validation and aborts the round on any signature, MAC, or timestamp failure.

The transport subsystem initializes two AEAD channels using RCS with KMAC authentication under the tunnel keys. Packet headers contain the message flag, sequence number, message length, and UTC timestamp. These fields are included as associated data in the AEAD transform. The implementation enforces strict replay resistance by verifying timestamp bounds and enforcing monotonic sequence numbers.

The cryptanalysis therefore focuses on five primary surfaces: the certificate hierarchy, topology propagation, MFK initialization, fragment generation and aggregation, and the

RCS-based tunnel. Each surface is examined in subsequent chapters for completeness, correctness, resistance to active attacks, adherence to the specification, and fidelity of the implementation to the formal security model.

Chapter 2: Problem Definition and Scope

2.1 Scope of the Protocol Under Analysis

This cryptanalysis concerns the **interior protocol MPDC-I**, defined in *MPDC-2024 Rev.1b*. The specification states explicitly that MPDC-I describes the complete set of interior-domain mechanisms necessary to establish an authenticated and encrypted duplex communication stream between a server (MAS) and a client. The protocol uses a multi-party distribution of entropy, authenticated topological control, and post-quantum cryptographic primitives defined in the referenced NIST standards.

The analysis includes all mechanisms that the specification designates as mandatory for a conformant MPDC-I deployment. These include:

- Certificate issuance, distribution, verification, and expiration.
- Topology convergence, incremental updates, and revocation propagation.
- Master fragment key establishment between MAS↔Agent and Client↔Agent.
- Fragment generation, fragment protection, MAS aggregation, and client verification.
- Session-key derivation and tunnel initialization.
- Tunnel encryption and authentication using RCS with per-packet sequence and timestamp validation.
- Error-handling rules, abort conditions, and replay-prevention logic.

All of these mechanisms are implemented in the provided source tree. Each one is examined in the cryptanalysis according to both the intended design and its realized behavior in the code.

2.2 Components Excluded from Analysis

The specification mentions MPDC-E, the exterior-domain protocol intended to interconnect multiple MPDC domains, authenticate external certificates, and support cross-domain key injection. This protocol is not defined in the Rev.1b specification and is therefore out of scope.

Other components the specification references but does not define, such as external governance procedures, administrative policy layers, or optional certificate-management extensions, are also excluded. The cryptanalysis does not consider any operational practices beyond those explicitly included in MPDC-I.

The reference implementation includes no placeholders for MPDC-E or for optional protocol extensions, and therefore no implicit surfaces exist that require separate treatment.

2.3 Assumptions Grounded in Cryptographic Primitives

MPDC-I depends directly on post-quantum primitives defined in NIST standards. The specification cites FIPS 202 for SHA-3 and SHAKE, SP 800-185 for cSHAKE and KMAC, FIPS 203 for ML-KEM (Kyber), and FIPS 204 for ML-DSA (Dilithium). The QSC library implements these primitives. The analysis assumes that these implementations behave as defined by their respective standards, and no claims are made beyond those standard definitions.

The cryptanalysis does not re-evaluate the security of the primitives themselves; it evaluates how MPDC-I composes them. All assertions of security in subsequent chapters are conditional on the standard assumptions for SHAKE, KMAC, Kyber, Dilithium, and the RCS AEAD construction.

2.4 Operational and Network Boundaries

The specification defines a fully adversarial network model in which an attacker controls all packet channels. Under this model an attacker may eavesdrop, drop, inject, modify, reorder, or replay packets. MPDC-I must therefore provide all authentication, confidentiality, and replay protection internally without relying on link-layer guarantees.

The protocol assumes that each participating device maintains a reasonably accurate system clock. Packet timestamps are validated against a maximum tolerance interval defined in *mpdc.h* as MPDC_PACKET_TIME_THRESHOLD, set in the implementation to sixty seconds. Although the specification presents a different example window, the implementation adopts the sixty-second constraint, and all cryptanalytic evaluation uses the actual compiled value.

Device compromise is treated according to the adversarial model in the specification: any subset of devices may be compromised before, during, or after protocol execution, except that the RDS private key is assumed uncompromised. The analysis evaluates MPDC-I's behavior under these partial-compromise conditions.

Network topology and certificate propagation rely entirely on cryptographically signed DLA broadcasts and incremental updates. No out-of-band trust channels or auxiliary validation authorities are assumed.

These boundaries define the complete environment within which the MPDC-I protocol is evaluated throughout this document.

Chapter 3: Model and Assumptions

3.1 Domain Architecture

The MPDC-I specification defines a hierarchical interior-domain architecture composed of five distinct device classes: the Root Domain Server (RDS), the Domain List Agent (DLA), the Management and Authentication Server (MAS), Clients, and Agents. Each device class has a separate cryptographic purpose, distinct certificate type, and defined interaction surface.

The **RDS** is the root trust anchor. It issues and signs all child certificates used by the DLA, MAS, Clients, and Agents. These certificates contain the device's signature verification key, issuer field, validity interval, configuration-set identifier, and a signature over the certificate hash. The specification treats the RDS key as uncompromised for the entire domain lifetime. The implementation enforces this through the functions in *certificate.c*, which require all certificates to verify under the RDS public key before acceptance.

The **DLA** distributes topological information, certificate updates, resignations, revocations, and convergence states. All DLA-issued messages are signed using Dilithium and include the timestamp and sequence number inside the signed hash input. The implementation performs these operations in *dla.c* and *network.c*, with strict rejection on any failed signature or time check.

The **MAS** serves as the key-derivation endpoint for the tunnel. It receives fragment ciphertexts from all participating Agents, authenticates and decrypts them using the MAS↔Agent master fragment key (mfk), generates its own MAS fragment, aggregates all contributions using a Keccak-based accumulator, and performs the tunnel-key derivation. The MAS also acts as the server endpoint in the encrypted data channel.

The **Client** interacts with the MAS to establish the tunnel. It receives the MAS-generated fragment bundle, verifies the authenticity of each fragment using the Client↔Agent mfks, decrypts its fragments, and reconstructs the same hash that seeds the tunnel keys.

Agents contribute entropy. Each Agent establishes a separate mfk with both the MAS and the Client. During a fragment-collection round, an Agent derives ephemeral fragment-encryption keys from its mfk, generates a fragment using SHAKE-256, encrypts and authenticates the fragment under the derived keys, and returns corresponding ciphertexts to the MAS.

The architecture therefore forms a directed trust structure with the RDS at the root, the DLA as the authenticated distributor of topology, MAS and Client as tunnel participants, and Agents as entropy contributors. The specification and implementation both adhere to this structure without deviation.

3.2 Roles, Authorities, and Certificate Hierarchy

Each device class operates under a certificate issued by the RDS. The certificate hierarchy is strictly enforced. Child certificates include:

- a Dilithium public verification key,
- an issuer string identifying the RDS,
- a serial number unique within the domain,
- a validity interval,
- a cryptographic configuration-set identifier,
- a designation field indicating device role,
- and a Dilithium signature binding all certificate fields.

The implementation validates certificates using the routines in *certificate.c*, which recompute the certificate hash, verify the Dilithium signature, verify issuer and designation fields, and check validity intervals. Devices never accept unsigned or self-signed certificates. MAS, Client, Agents, and DLA all perform these verifications prior to any state update.

Topology events, including incremental updates, convergence responses, and status messages, are signed under the DLA certificate or under the device's own certificate, depending on the event type. The sub-header containing the packet's sequence and timestamp is always included in the signed region. The implementation validates this through network_certificate_signed_hash_verify and related functions in *network.c*. Any mismatch results in immediate rejection.

The certificate hierarchy therefore functions exactly as described in the specification: a single-root PKI with role-specific child certificates and strict verification at each hop.

3.3 Network Assumptions

MPDC-I assumes a fully adversarial network. The specification states that an attacker may observe, modify, inject, drop, reorder, or replay any message exchanged between devices. No link-layer security assumptions are made. All cryptographic protections must be provided by the MPDC protocol itself.

Time is a required component of all messages. Each packet contains a 64-bit UTC timestamp. Every device must maintain a clock sufficiently synchronized that incoming packets fall within the permitted time window. The implementation enforces this using mpdc_packet_time_valid, which checks that the timestamp difference does not exceed the configured MPDC_PACKET_TIME_THRESHOLD. In the provided implementation this threshold is sixty seconds.

The network is assumed to provide eventual connectivity. If a device does not receive topology broadcasts or certificate updates, it will not accept stale information and will refuse communications. This is an intentional design constraint. The specification identifies this as part of MPDC-I's security posture, preferring fail-closed behavior over incorrect acceptance of stale or replayed data.

Sequence numbers are strictly monotonic per direction. Out-of-order messages are rejected. Packets are not accepted if they repeat a sequence number or if the expected increment does not match. This behavior appears in the decrypt pipeline in *mpdc.c* and is required for replay resistance.

These assumptions are consistent across the specification and the implementation.

3.4 Time, Synchronization, and Replay Tolerance

Time and replay control are central to MPDC-I. The specification states that every control-plane message includes a timestamp and a sequence number, and that both values must be included in the signed or MAC-authenticated portion of the message. The implementation serializes the sequence and timestamp into the sub-header and hashes them together with the message body before signature or MAC operations.

The timestamp tolerance window restricts packet acceptance to those arriving within the configured threshold. Replay of older packets is prevented because the timestamp in the signed or MAC-authenticated region will fall outside the permitted window. The implementation enforces this rule consistently, and no code path bypasses the time check.

Sequence numbers provide a second layer of replay and ordering protection. Each connection direction maintains an independent 64-bit counter. Transmitters increment their counters monotonically; receivers verify that the incoming packet sequence matches exactly the expected value. The implementation rejects any packet with an incorrect sequence value. This blocks replay of previously valid packets, reflection of traffic from one direction to the other, and any attempt to reorder traffic.

Together, timestamp validation and sequence enforcement provide strong replay protection in accordance with the specification. The implementation matches this design without deviation.

Chapter 4: Related Work and Context

4.1 Cryptographic Domain and Notation

MPDC-I is situated within the domain of multi-party authenticated key establishment and symmetric-tunnel construction. The protocol uses a combination of post-quantum asymmetric primitives (ML-KEM Kyber and ML-DSA Dilithium), Keccak-based KDF and MAC functions (SHAKE, cSHAKE, KMAC), and an AEAD stream cipher (RCS). All primitives are standard and defined in FIPS 202, FIPS 203, FIPS 204, and SP 800-185, and the implementation adheres to the input and output conventions of these standards through the QSC library. No notation in MPDC-I contradicts these underlying standards.

All variables used in the analysis that represent keys, hashes, fragments, or ciphertexts map to the fields and structures defined in *mpdc.h* and the associated code paths.

4.2 Background on Post-Quantum Components

The cryptographic components used in MPDC-I align with current NIST standardization. Kyber provides IND-CCA-secure KEM operations used for the establishment of master fragment keys. Dilithium provides EUF-CMA security for certificates, topology propagation, and authenticated network messages. SHAKE-256 provides an extendable-output source for fragment generation and tunnel-key derivation. cSHAKE and KMAC are used for domain-separated symmetric-key derivation and message authentication.

These primitives collectively support MPDC-I's requirement that the protocol remain secure even after the hypothetical emergence of large-scale quantum computers. Their use is consistent across the specification and the C implementation.

4.3 Context in Domain-Based Network Security

Traditional domain-based network-security architectures rely on certificate hierarchies and single-endpoint AKE protocols. MPDC-I extends this by distributing the entropy of the session key across multiple authenticated Agents within the domain. This approach incorporates aspects of hierarchical PKI, multi-device coordination, and symmetric-tunnel establishment. The RDS anchor, DLA control and distribution mechanism, and the MAS-Client tunnel are all part of this controlled interior-domain architecture. The aim is to reduce reliance on a single handshake or single device for session-key derivation.

In this sense, MPDC-I belongs to a class of protocols designed for controlled network environments in which multiple authenticated devices contribute to trust and entropy, but where only two primary devices (MAS and client) ultimately derive the tunnel keys.

4.4 Previous Work in Multi-Agent Entropy Distribution

Few deployed protocols distribute entropy across multiple independent entities. Some distributed randomness-generation systems combine entropy from semi-trusted contributors, but these are often tied to threshold cryptosystems or decentralized ledgers rather than secure tunnel establishment. Other multi-party constructions, such as group key agreement protocols, require that all participants derive the same shared key, which is not the case in MPDC-I.

The specification explicitly states that MPDC-I is not a group key-agreement protocol and that Agents do not derive the final session keys. Instead, Agents provide independently generated fragments that the MAS and client use to construct a tunnel key known only to these two endpoints. This design is unique compared to classical AKE frameworks in that it creates a two-party tunnel whose entropy is sourced from a multi-party environment.

4.5 Distinguishing Characteristics of MPDC-I

MPDC-I differs from conventional key-exchange protocols in several ways that are central to the analysis performed in this document.

First, the session keys are not derived from a single public-key handshake, but from a collection of short pseudo-random fragments produced by authenticated Agents. Second, the trust architecture enforces the use of RDS-signed certificates and DLA-signed topology updates that incorporate timestamps and sequence numbers, preventing replay and certificate downgrades. Third, the MAS contributes its own fragment to the session, and both MAS and client perform identical aggregation operations to derive the tunnel key. Fourth, the data path uses a Keccak-based AEAD cipher and binds packet metadata tightly to the authenticated encryption layer. These properties cause MPDC-I to behave differently from classical two-party designs and motivate the cryptanalytic questions addressed in later chapters.

4.6 Analytical Position

The analysis in the remaining chapters evaluates MPDC-I in reference to the properties enumerated in the specification: entity authentication, session-key secrecy, forward secrecy, predictive resistance, replay resistance, downgrade resistance, and robustness. Each mechanism in the protocol is examined in the context of these properties. The evaluation considers both the formal behavior described in the specification and the realized behavior in the implementation.

This document does not assume any properties not explicitly stated in the specification or demonstrated in the code. Each conclusion is derived from either standardized cryptographic assumptions or observable behavior in the provided implementation.

Chapter 5: Preliminaries

5.1 Notational Conventions

This document uses notation consistent with the MPDC-I specification and the variables present in the implementation. Lowercase italic letters denote scalar values such as timestamps *t*, sequence numbers *s*, or counters *c*. Uppercase italic letters denote structured data such as certificates *C*, messages *M*, or keys *K*. Vectors such as fragment ciphertext pairs use boldface sequences in prose, but the underlying data structures correspond directly to the arrays defined in *mpdc.h* and *network.c*. All cryptographic functions are written in standard form, for example SHAKE256, KMAC, or ML-KEM, without nonstandard symbols.

Where notation appears in the specification, such as hashed certificate fields or concatenated message structures, it is mapped directly to the serialized forms implemented in *network.c*, *certificate.c*, and *mpdc.c*. No alternative notation is introduced.

5.2 Cryptographic Primitives Used in MPDC-I

MPDC-I uses a fixed set of cryptographic primitives defined in NIST standards. Certificate signatures use ML-DSA (Dilithium), and key-encapsulation operations for master fragment keys use ML-KEM (Kyber). Optional alternative configurations can use combinations of McEliece and ML-KEM or McEliece and SPHINCS+ (SLH-DSA). Parameter sets are defined in the adjoining QSC cryptographic library for each of the asymmetric ciphers and signature schemes.

Fragment generation, fragment-encryption key derivation, and tunnel initialization use SHAKE-256 or cSHAKE with domain-separated input as defined in SP 800-185. MAC operations on fragments use KMAC with keys derived from ephemeral fragment-encryption keys. Tunnel encryption uses the RCS authenticated cipher (or optionally AES-GCM), which uses a Keccak sponge to generate keystream and KMAC-derived tags.

The implementation uses the QSC library directly for all of these primitives. No wrappers, modifications, or protocol-level adaptations alter the expected input or output structure of these functions. All cryptographic semantics therefore match the referenced standards.

5.3 Packet Structure and Sub-Header Binding

Each MPDC packet consists of a 22-byte header followed by a message payload. The header contains the packet flag, message length, sequence number, and UTC timestamp. These fields are defined in *mpdc.h* and serialized by routines in *mpdc.c*.

The specification states that the sub-header, consisting of the serialized sequence number and timestamp, must be hashed into every signature or MAC operation covering control-plane messages. The implementation follows this by constructing the sub-header explicitly and including it in the hash passed to the signature or MAC functions. Tunnel packets bind the entire header as associated data into the RCS cipher. This ensures that none of these fields can be altered without detection.

These structures are central to MPDC-I's replay and downgrade resistance and are part of the formal definitions examined later.

5.4 Certificate Structures and Field Semantics

Certificates in MPDC-I contain fields for the device's Dilithium public key, issuer, expiration interval, serial number, cryptographic configuration identifier, and designation. The child certificates are signed by the RDS. The specification defines these fields in normative form, and the implementation stores them in the mpdc_certificate_child structure.

The certificate signature covers a SHA-3 hash over the serialized certificate body. The implementation recomputes this hash and verifies the signature before accepting any certificate. The semantics of these fields, including validity windows and configuration identifiers, are later used in the formal evaluation of authenticity and downgrade resistance.

5.5 Master Fragment Keys and Derivation Context

The master fragment key mfk is a shared secret established between an Agent and the MAS, and separately between an Agent and the Client. These keys are generated using Kyber encapsulation and decapsulation during the MFK exchange defined in the specification. Once established, the mfk forms the root for all fragment-encryption keys efk and all MAC keys used in fragment authentication.

The implementation derives per-round efk values by hashing together the mfk with the certificate-hash values corresponding to the participants and with the MAS-supplied token for the round. These derivations are implemented using Keccak-based functions consistent with the specification.

Understanding how these keys are derived and used is essential for later analysis of confidentiality, predictive resistance, and forward secrecy.

5.6 Fragment Construction and Ephemeral Keys

A fragment consists of a 256-bit value generated via SHAKE-256, masked under an efk derived from the mfk, and authenticated using a KMAC tag derived from the same efk. Each Agent generates two such ciphertexts for each fragment: one intended for the MAS and one for the Client. These structures correspond to the fields in the mpdc_fragment_query_response structures in *network.c*.

Fragments are strictly ephemeral. After a fragment-collection round completes, the fragment values, the efk, and any per-round metadata used in their derivation are not used in subsequent rounds. This property is important for predictive-resistance arguments, which assume that future efk values cannot be computed from previously observed state.

5.7 Tunnel-Key Derivation and Cipher Initialization

Once the MAS and the Client have processed the same set of fragments and validated their authenticity using the appropriate MAC keys, both sides compute a Keccak-based hash over the ordered fragment sequence and the MAS contribution. This produces a 256-bit value denoted *h* in the specification. This value is expanded into transmit and receive keys for the RCS cipher.

The implementation performs this key expansion through SHAKE-256 and initializes separate RCS states for each communication direction. Sequence counters are initialized to zero. This step marks the transition from the control-plane establishment process to the encrypted data-plane.

5.8 Replay Window and Temporal Constraints

MPDC-I incorporates time into both control-plane and data-plane message validation. The timestamp included in each header is compared against the device’s local time using the threshold value defined in MPDC_PACKET_TIME_THRESHOLD. This threshold, set to sixty seconds in the implementation, bounds the acceptable time difference for all messages.

Because the sub-header binds the sequence number and timestamp together into signatures and MACs, any replayed message with a timestamp falling outside the window will be rejected before authentication, and any replay using a valid timestamp but mismatched sequence will be rejected due to sequence monotonicity.

Understanding this temporal model is essential for analyzing replay, reflection, and downgrade resistance in later chapters.

Chapter 6: Protocol Overview

6.1 Architectural Summary

MPDC-I operates through a coordinated set of devices under a hierarchical trust model anchored by the Root Domain Server. The Domain List Agent distributes signed topology information, certificate updates, and revocation events. Clients and the Management and Authentication Server establish and maintain authenticated relationships with Agents. These relationships are used to generate per-Agent master fragment keys and to collect pseudo-random fragments that serve as inputs to the final tunnel-key derivation. The tunnel itself uses the RCS authenticated encryption scheme, binding packet metadata into the AEAD layer with strict timestamp and sequence enforcement.

The protocol proceeds through distinct operational phases: certificate acquisition, topology initialization, master fragment key establishment, fragment collection, key derivation, and encrypted-channel operation. Each phase has explicit checks for identity, integrity, freshness, and replay resistance.

6.2 Certificate and Topology Initialization

Every device in the MPDC domain possesses an RDS-signed certificate containing its post-quantum public key, device designation, issuer information, validity interval, and configuration identifier. Certificate verification is performed before any device participates in the protocol.

After certificate validation, a device performs a registration exchange with the DLA. The DLA returns its own certificate and distributes the initial topology, a signed list of known MAS servers and Agents. Incremental updates are requested and received through signed messages that include the packet’s timestamp and sequence number in the authenticated region.

Topology convergence messages distribute new or updated certificates, revocation notices, and reorganized lists of domain participants. Both the specification and the implementation require the DLA signature to validate, and any mismatch or out-of-window timestamp forces rejection.

6.3 Master Fragment Key Establishment

A master fragment key is a long-term shared secret between each Agent and the MAS, and between each Agent and the Client. These keys are established through authenticated Kyber encapsulation. The exchange is structured so that:

1. The initiator sends an authenticated request containing its child certificate and a signature over the sequence and timestamp of the request.
2. The responder generates a Kyber public key, signs it together with the sub-header, and returns it.
3. The initiator encapsulates, signs the ciphertext, and sends the establish message.
4. The responder verifies the signature, decapsulates, and obtains the same master fragment key.

These keys persist until certificate expiration and form the root secrets for deriving all ephemeral fragment-encryption keys used in fragment-collection rounds.

6.4 Fragment Generation and Distribution

Fragments are generated by Agents in response to MAS-initiated collection requests. The MAS sends a request containing the Client’s serial number and token. For each Agent, the MAS derives a MAC key from its MFK with the Client and authenticates the collection query.

An Agent verifies the MAC, derives two ephemeral fragment-encryption keys from its MFKs with the MAS and Client, and generates a fragment using SHAKE-256. The fragment is separately masked under each efk and authenticated with a KMAC tag derived from each efk. The Agent returns a message containing:

- its serial number,

- the fragment ciphertext for the MAS,
- the fragment ciphertext intended for the Client,
- and MACs over the protected fields and sub-header.

The MAS receives all authenticated Agent responses during the round.

6.5 MAS Aggregation and Client Verification

After collecting responses, the MAS decrypts each MAS-side fragment, verifies all MACs, and updates a Keccak accumulator with each recovered fragment. MAS also adds its own MAS fragment into the same accumulator.

The MAS then prepares the Client response. It encrypts the MAS fragment under the MAS↔Client MFK derivation, appends the Client-side ciphertexts for each Agent, and authenticates the message with a KMAC tag tied to the sub-header. The Client receives this message, verifies all MACs, derives and decrypts each Agent fragment using its own MFKs, and reconstructs the same accumulated value.

This accumulated hash value becomes the basis for the tunnel key.

6.6 Tunnel-Key Derivation and Cipher Initialization

Both MAS and Client compute the hash value *h* representing the aggregation of all authenticated fragments. This value is expanded via SHAKE-256 into transmit and receive keys for the RCS cipher. The implementation initializes independent cipher states for each direction and sets both sequence counters to zero.

From this point onward, all communication is conducted within an authenticated and encrypted channel. The control plane does not participate further except for topology updates and revocation reception.

6.7 Authenticated Transport and Replay Resistance

Each tunnel packet contains a header with the packet flag, message length, sequence number, and UTC timestamp. The RCS cipher authenticates the entire header as associated data, binding these fields into each ciphertext. The decrypt routine enforces:

- sequence equality with the expected counter,
- timestamp within the permitted threshold,

- correct RCS authentication tag,
- and valid message length.

Any discrepancy results in immediate rejection and session termination.

This transport layer provides confidentiality, integrity, ordering guarantee, and replay resistance as described in the specification and implemented in the transport functions of *mpdc.c*.

6.8 Revocation, Topology Change, and Fail-Closed Behavior

The DLA distributes revocation messages and certificate updates through signed convergence broadcasts and incremental updates. Devices invalidate local topology entries that fail signature or timestamp verification. Revocation of an Agent, MAS, or Client certificate results in immediate discarding of the corresponding relationships, including master fragment keys and cached fragments.

The implementation is designed to fail closed: any failed signature, MAC, timestamp, or sequence validation aborts the affected run. This behavior conforms to the robustness requirement described in the MPDC-I specification.

Chapter 7: Internal Functions

7.1 Certificate-Processing Functions

MPDC-I defines a set of internal routines responsible for parsing, verifying, and reconstructing certificates. These operations ensure that all certificate inputs and topology messages are validated before a device transitions to a dependent state.
The implementation parses certificate structures using the routines in *certificate.c*, including functions that deserialize fields, recompute the certificate hash using SHA3, and verify the signature using the Dilithium verification key of the RDS. The certificate fields include the device designation, configuration-set identifier, serial number, validity interval, public verification key, and issuer string. Internal functions validate each of these fields before the certificate is accepted. Time validity is checked by comparing the certificate's validity interval with the local clock.
Topology synchronization relies on similar routines where the certificate-hash fields inside topology messages are verified using the DLA's public verification key. These

functions ensure that devices only accept topology information anchored at legitimate certificates.

7.2 Sub-Header and Signed-Message Functions

Every authenticated control-plane message in MPDC-I uses a signed or MAC-protected data structure that includes the serialized sequence number and UTC timestamp. The internal routines responsible for constructing and verifying these structures appear in *network.c*.

Packet header serialization, performed by mpdc_packet_header_serialize in *mpdc.c*, yields the full 22-byte header. The sub-header, composed of the sequence number and timestamp, is extracted from this header and inserted into a signed-message structure prior to hashing.

Verification functions reconstruct the same sub-header from the received header, recompute the signed hash, and check the signature or MAC using the appropriate key. These routines ensure that the message body cannot be separated from its freshness parameters, enforcing the protocol's replay bounds.

Internal checks abort execution on any mismatch, ensuring that no later function processes unverified message contents.

7.3 Master Fragment Key Establishment Functions

The functions implementing the MFK establishment exchange combine Kyber operations with Dilithium verification and sub-header binding. These functions appear across *network.c*, *mas.c*, *client.c*, and *agent.c*.

First, the MFK request message uses network_mfk_exchange_request_packet to populate fields, attach signatures, and construct the packet. The response is built by network_mfk_exchange_response_packet, which incorporates a new Kyber public key and the responder's certificate verification key, bound by signature.

The establish message uses encapsulation to produce the final ciphertext and shared secret. The corresponding verification functions reconstruct the sub-header, verify the signature, and invoke the Kyber decapsulation function to complete the MFK.

Internally, each MFK is stored in the device's state table, keyed by the agent serial or MAS/client serial. These persistent keys are used by the fragment-derivation functions described below.

7.4 Fragment-Derivation and Fragment-Protection Functions

Fragment-encryption keys, denoted efk, and fragment MAC keys are derived from the master fragment key using SHAKE or cSHAKE. These routines appear in *network.c* and *agent.c* as network_derive_fkey and related helper functions.
Each efk is a function of the master fragment key, certificate hash values of the involved parties, and the MAS token for the round. These values are concatenated and hashed, producing a key that is unique for every agent-MAS-client combination and every fragment-collection round.
Fragment generation uses SHAKE-256 to produce a pseudo-random 256-bit fragment. This fragment is XOR-masked with the efk for confidentiality, generating two ciphertexts, one for the MAS and one for the client. Fragment MAC tags are produced using KMAC with a key derived from the same efk. The internal functions responsible for these operations enforce that the plaintext fragment never appears on the wire.
Verification functions in MAS and client derive the same efk, recompute the MACs, and recover the fragment if and only if MAC verification succeeds. Internal error-handling paths ensure that any failed MAC invalidates the entire fragment round.

7.5 Fragment Aggregation and Hashing Functions

The MAS uses its internal aggregation functions to combine validated fragments into a single hash value. The primary function performing this aggregation is implemented in *network.c*, where the MAS processes each decoded fragment, updates a Keccak sponge state, and includes the MAS fragment.
These functions treat fragment ordering deterministically. Each fragment is processed exactly once, and all fragments included in the MAS response to the client correspond to authenticated Agent responses for that round.
The client uses a mirror function, reconstructing the same fragment accumulator from the ciphertexts and verifying that its locally computed hash equals the MAS-derived value. The internal implementation enforces strict equality and aborts if mismatch occurs.

7.6 Tunnel-Key Expansion and State Initialization Functions

The tunnel key is produced by expanding the final Keccak-derived hash value *h* into two symmetric keys through a SHAKE-based KDF. These keys initialize the RCS cipher states for the transmit and receive channels.
Internal functions allocate new cipher-state structures, set the key material, reset sequence counters to zero, and transition the device to the established state. These

functions appear in *mpdc.c* and are triggered immediately after successful fragment aggregation and hash comparison.

7.7 Transport-Layer Functions and AEAD Enforcement

The internal functions for packet construction and decryption appear in *mpdc.c*. Packet creation sets the flag, length, sequence number, and timestamp, serializes the header, and passes the header as associated data to the RCS cipher.
Decryption verifies the timestamp using mpdc_packet_time_valid, verifies the sequence number, checks the RCS authentication tag, and deserializes the payload. No packet is accepted if any of these checks fail.
These internal functions form the foundation of the MPDC-I replay and ordering enforcement model and directly implement the specification's requirement that packet metadata be authenticated.

7.8 Revocation and Topology-Maintenance Functions

The DLA uses internal certificate-hash broadcast routines implemented in *dla.c* to propagate topological changes and revocation events. Devices receive these messages and verify the associated signatures using functions in *network.c*. Valid updates cause the device to recompute certificate tables and update internal structures tracking MAS and Agent availability.
These functions ensure that devices reject stale or revoked certificates and adjust fragment relationships accordingly. The implementation completely prohibits use of invalidated certificate entries, preserving the fail-closed behavior required by the specification.

Chapter 8: Performance and Operational Behavior

8.1 Computational Characteristics of Core Operations

MPDC-I is organized so that the most computationally intensive operations occur outside the steady-state data path. Certificate verification and authenticated KEM exchanges use Dilithium and Kyber, which require nontrivial polynomial and lattice-based computations. These operations occur during certificate acquisition, topology synchronization, and master fragment key establishment, all of which take place before the encrypted tunnel is created.

The steady-state operations that occur during fragment-collection rounds and tunnel maintenance rely primarily on symmetric Keccak-based functions. SHAKE-256, cSHAKE, and KMAC operations dominate fragment generation, fragment-encryption key derivation, and fragment MAC verification. These functions are computationally efficient relative to the post-quantum primitives and impose predictable cost per fragment.

The RCS AEAD cipher uses a Keccak sponge internally and performs authenticated encryption with a single associated-data binding step per packet. The per-packet cost consists of header serialization, state absorption of the header, symmetric transform of the payload, and tag generation. This is significantly less costly than classical block-cipher modes or hybrid PQC operations and is suitable for high-throughput communication.

8.2 Certificate and Topology Propagation Overhead

Certificate propagation and topology synchronization are infrequent and occur primarily during initialization or when the DLA receives updated certificate lists. These operations involve signature verification of DLA broadcasts and certificate-hash updates. Because the broadcast messages include Dilithium signatures and certificate structures, verification costs are dominated by Dilithium signature verification and hash recomputation.

The implementation performs these checks once per update event. Devices cache valid certificates and only invalidate or refresh them when the DLA distributes new topology messages. As a result, the computational overhead of certificate and topology maintenance remains low in steady state.

8.3 Fragment-Collection Performance

Fragment generation and protection impose linear cost with respect to the number of participating Agents. Each Agent performs two efk derivations and generates one 256-bit fragment per round. The MAS processes each Agent response by verifying two MAC tags, decrypting the MAS-side fragment, and updating the Keccak accumulator. The client repeats the same operations using its own keys.

The cost per fragment is dominated by two KMAC verifications, one XOR-decryption of the masked fragment, and one Keccak absorb operation. These are symmetric operations and scale predictably with the number of Agents. The implementation does not introduce branching or variable-time execution in this loop, and the cost increases linearly with the domain’s size.

Because fragment rounds occur only at tunnel establishment or periodic rekeying, they do not affect the steady-state cost of encrypted data transfer.

8.4 Tunnel Establishment and Cryptographic State Initialization

Tunnel establishment occurs after successful fragment aggregation. The cost consists of two SHAKE expansion operations to derive transmit and receive keys for the RCS cipher. Initialization of RCS state structures and sequence counters is minimal.

The initialization cost is therefore modest relative to the cost of the KEM operations and certificate verification performed earlier in the protocol. The implementation initializes separate transmit and receive channels, ensuring unidirectional independence of the symmetric keys.

8.5 Data-Plane Throughput

The RCS-based transport layer experiences symmetric overhead per packet. Each packet requires:

1. serialization of the fixed 22-byte header,
2. absorbing the header as associated data into the RCS state,
3. generating a keystream for the payload length,
4. XOR-transforming the payload, and
5. producing or verifying a KMAC-derived authentication tag.

Because Keccak permutations are computationally efficient, the cost per byte remains low. The implementation does not allocate or free memory during packet processing except for the buffers required to hold the packet, and the packet format remains fixed throughout the session. These properties enable uniform throughput characteristics independent of message structure.

Sequence-number and timestamp checks impose negligible overhead relative to the symmetric transformations. Both fields are read from the header and compared in constant time.

8.6 Topology Convergence and Update Latency

Topology convergence depends primarily on the DLA's broadcast interval and the number of devices in the domain. Upon receiving a topology broadcast, devices verify the DLA signature, parse certificate entries, and update internal structures accordingly. The latency of topology propagation is dominated by broadcast frequency and network characteristics, not cryptographic operations.

Incremental updates require a request-response exchange and introduce additional Dilithium verification. These operations remain infrequent relative to data-plane traffic and do not contribute meaningfully to runtime overhead.

8.7 Revocation and Recovery Behavior

Revocation events occur when the DLA distributes updated certificate hashes indicating that a certificate has been revoked. Devices receiving such broadcasts verify the DLA signature, invalidate local records of the revoked device, and remove any cached master fragment keys associated with the device.

This process involves hash comparisons and small table updates. There are no heavy cryptographic operations other than signature verification of the broadcast. The protocol is designed to fail closed on any inconsistency, and devices do not attempt to continue using stale keys or invalidated associations.

8.8 Reliability and Fail-Closed Operation

MPDC-I is built to fail closed on any signature mismatch, MAC failure, timestamp violation, sequence error, or certificate inconsistency. This behavior ensures that no partial or degraded communication state can occur. The consequence is that reliability depends strongly on correct time maintenance, accurate sequence counters, and availability of valid topology broadcasts.

Operationally, this model prioritizes cryptographic integrity over message availability. Lost or out-of-order packets cannot be recovered; instead, the session is terminated and must be reestablished. This behavior is consistent with the specification and does not introduce ambiguity into the protocol's security model.

Chapter 9: Mathematical Description

9.1 Notation and Domain Model

Let *C* be a child certificate.
Its certificate-hash value is:

$h\_C = \mathrm{SHA3}(C\_body)$

Verification checks the Dilithium (ML-DSA) signature:

$\mathrm{Verify}(vk\_RDS, h\_C, \sigma\_C) = \mathrm{true}$

Each MPDC packet header is defined as:

$H = (f, \ell, s, t)$

For signed and MAC-protected messages, the sub-header is:

$SH = (s, t)$

This sub-header is always included in the authenticated data.

9.2 Certificate Structure and Signed-Message Binding

The signed hash of a control-plane message *M* is:

$h\_M = \mathrm{SHA3}(M \parallel s \parallel t)$

Signing operation:

$\sigma\_M = \mathrm{Sign}(sk, h\_M)$

Verification reconstructs the same hash:

$h\_M' = \mathrm{SHA3}(M \parallel s \parallel t)$

and checks:

$\mathrm{Verify}(vk, h\_M', \sigma\_M) = \mathrm{true}$

Any modification of *s* or *t* invalidates the signature.

9.3 Master Fragment Key Exchange

In the authenticated Kyber exchange, the initiator and responder derive the shared secret:

$(ct, k) = \mathrm{Encaps}(pk\_R)$

$k' = \mathrm{Decaps}(ct, sk\_R)$

If all signatures verify and $k = k'$, the resulting master fragment key is:

$\text{mfk} = k$

Each Agent holds:

- mfk_A_M (Agent ↔ MAS)
- mfk_A_C (Agent ↔ Client)

These are long-term secrets used for fragment-key derivation.

9.4 Fragment Generation and Encryption Keys

Each Agent generates a 256-bit fragment *F* using the QSC random generator.

MAS-side ephemeral key

$\text{efk\_A}{\rightarrow}\text{MAS} = \text{SHA3}(\text{mfk\_A\_M} \parallel \text{h\_A} \parallel \text{h\_MAS} \parallel \tau)$

Client-side ephemeral key

$\text{efk\_A}{\rightarrow}\text{Client} = \text{SHA3}(\text{mfk\_A\_C} \parallel \text{h\_A} \parallel \text{h\_Client} \parallel \tau)$

Fragment masking

$\text{C\_MAS} = \text{F XOR efk\_A}{\rightarrow}\text{MAS}$

$\text{C\_Client} = \text{F XOR efk\_A}{\rightarrow}\text{Client}$

Fragment MAC tags

MAS-side:

$\text{tag\_MAS} = \text{KMAC}(\text{efk\_A}{\rightarrow}\text{MAS}\,, \text{C\_MAS} \parallel \text{SH})$

Client-side:

$\text{tag\_Client} = \text{KMAC}(\text{efk\_A}{\rightarrow}\text{Client}\,, \text{C\_Client} \parallel \text{SH})$

Only a party possessing the correct mfk and the current τ can recompute these values.

9.5 MAS Aggregation Process

After MAC verification and decryption of Agent fragments:

$\text{F\_i} = \text{C\_i\_MAS XOR efk\_i}{\rightarrow}\text{MAS}$

The MAS absorbs each fragment into a Keccak state *S*:

$\text{S} \leftarrow \text{Keccak.Absorb}(\text{S, F\_i})$

The MAS generates its own fragment $\mathrm{F\_MAS}$ and absorbs it:

$\mathrm{S} \leftarrow \mathrm{Keccak.Absorb(S, F\_MAS)}$

The final aggregated value is:

$\mathrm{h} = \mathrm{Keccak.Final(S)}$

The Client performs the same operations using $\mathrm{C\_i\_Client}$ and $\mathrm{efk\_i{\rightarrow}Client}$, producing an identical h only if all authentication checks succeed.

9.6 Tunnel-Key Derivation

Two independent tunnel keys are derived from *h*:

$\mathrm{K\_tx} = \mathrm{SHAKE256(h \parallel 01)}$

$\mathrm{K\_rx} = \mathrm{SHAKE256(h \parallel 02)}$

These initialize the transmit and receive cipher states.

9.7 Packet Authentication Model

For each tunnel packet:

- header H is associated data,
- payload P is encrypted,
- ciphertext C and tag tag are produced by the cipher.

Encryption:

$\mathrm{(C, tag)} = \mathrm{Enc(K, H, P)}$

Decryption (accept only if authentication succeeds):

$\mathrm{P} = \mathrm{Dec(K, H, C, tag)}$

Since H contains (s, t), sequence and time are cryptographically bound to each packet.

9.8 Replay Acceptance Conditions

A received packet with header $(\mathrm{f},\ell,\mathrm{s},\mathrm{t})$ is accepted only if all hold:

$|\mathrm{t} - \mathrm{t\_local}| \leq \mathrm{T\_max}$

$\mathrm{s} = \mathrm{s\_expected}$

tag verifies under the active cipher state

In the reference implementation, $\mathrm{T\_max} = 60\ \mathrm{seconds}$ by default.

9.9 Security Invariants

The protocol enforces the following properties:

1. All certificates must validate against the RDS root.
2. The sub-header *(s, t)* used in authentication matches the received header.
3. Only fragments authenticated with the correct efk are accepted.
4. MAS and Client must compute identical h before tunnel establishment.
5. Tunnel packets are accepted only if sequence, timestamp, and AEAD tag checks succeed.
6. Session keys differ each round because h depends on ephemeral fragments and the MAS token τ.

These invariants hold in both the specification and the implementation.

Chapter 10: Security Analysis

10.1 Certificate-Chain Security

MPDC-I depends on the authenticity of RDS-signed certificates.
Each certificate is validated by recomputing its certificate hash and verifying the ML-DSA signature.
A certificate is accepted only if:

$$\mathrm{Verify}(\mathrm{vk\_RDS}\,,\, \mathrm{h\_C}\,,\, \sigma\mathrm{\_C}) = \mathrm{true}$$

The implementation performs this check in every certificate-processing path.
Because the DLA cannot forge RDS signatures, impersonation attacks reduce to the hardness of ML-DSA.
Topology broadcasts, resignation messages, and revocation updates embed the timestamp and sequence number into their signed data, ensuring that stale or replayed messages fail validation.

This aligns completely with the specification.

10.2 Authentication and Freshness Binding

All authenticated MPDC messages bind the sub-header in the signed hash:

$$h\_M = SHA3(M \parallel s \parallel t)$$

Thus any modification to *s* or *t* invalidates the signature.
The receiver reconstructs the authenticated value:

$$h\_M' = SHA3(M \parallel s \parallel t)$$

and accepts the message only if:

$$Verify(vk , h\_M' , \sigma\_M) = true$$

This simultaneously provides entity authentication and freshness.
Replay is prevented because an attacker cannot produce a valid signature over a modified timestamp or sequence value.

10.3 Security of Master Fragment Key Establishment

The MFK exchange is an authenticated ML-KEM flow.
The shared key must satisfy:

$$(ct , k) = Encaps(pk\_R)$$
$$k' = Decaps(ct , sk\_R)$$
$$k = k'$$

This equality holds only if the ciphertext is valid under the responder's key.
Because all MFK messages include signatures over *(s, t)* and the Kyber public key, acceptance requires:

$$Verify(vk , SHA3(M \parallel s \parallel t) , \sigma\_M) = true$$

Any mismatch aborts the exchange.
Thus, the only way to learn *mfk* is to break ML-KEM or forge ML-DSA signatures.
The implementation conforms exactly to these conditions.

10.4 Fragment Confidentiality and Integrity

Fragment ciphertexts are produced as:

$C\_MAS = F \text{ XOR } efk\_A \rightarrow MAS$
$C\_Client = F \text{ XOR } efk\_A \rightarrow Client$

And MAC tags must satisfy:

$tag\_MAS = KMAC(efk\_A \rightarrow MAS , C\_MAS \parallel SH)$
$tag\_Client = KMAC(efk\_A \rightarrow Client , C\_Client \parallel SH)$

A fragment response is valid only if both:

$VerifyMAC(tag\_MAS) = true$
$VerifyMAC(tag\_Client) = true$

Because *efk* is derived as:

$efk = SHA3(mfk \parallel h\_A \parallel h\_X \parallel \tau)$

an attacker who does not know *mfk* cannot compute *efk*.
The use of the MAS token *τ* ensures that efk values are fresh each round, which prevents prediction of future fragments even if previous fragments were observed.

10.5 Correctness and Security of MAS Aggregation

The MAS reconstructs each fragment:

$F\_i = C\_i\_MAS \text{ XOR } efk\_i \rightarrow MAS$

Each reconstructed fragment is added to the accumulator:

$S \leftarrow Keccak.Absorb(S , F\_i)$

and the MAS fragment contributes:

$S \leftarrow Keccak.Absorb(S , F\_MAS)$

The final aggregated value is:

$h = Keccak.Final(S)$

The Client performs symmetric operations and must compute exactly the same:

$h\_Client = h\_MAS$

Any mismatch terminates the session.
Thus, *h* is a function of:

- all authenticated Agent fragments

- the MAS fragment
- the MAS token
- all participating certificate hashes
- all MFK values

No attacker can influence *h* without forging MACs or signatures.

10.6 Tunnel-Key Secrecy

Tunnel keys are derived from *h*:

$K\_tx = SHAKE256(h \parallel 01)$
$K\_rx = SHAKE256(h \parallel 02)$

An attacker must therefore reconstruct *h* to break confidentiality.
To compute *h*, the attacker must recover all contributing fragments.
Recovering fragments requires recovering all corresponding *efk* values, which requires all corresponding *mfk* values.

Thus, secrecy of the tunnel depends on:

At least one honest Agent per round

If even one Agent remains uncompromised, the attacker cannot derive *h*, since *F* is 256-bit high entropy.

10.7 Forward Secrecy and Post-Compromise Properties

Forward secrecy results from the use of ephemeral fragments and fresh efk keys.
Even if an attacker compromises *mfk* after a session has completed, reconstruction of past fragments requires knowing the historical MAS token τ and all ephemeral efk inputs, which are not stored.

Thus the attacker cannot reconstruct past:

F_i

nor the past aggregated value:

h_past

For future rounds, predictive resistance follows from the requirement that τ be generated by the MAS after the previous fragment round:

$$\mathrm{efk\_future} = \mathrm{SHA3}(\mathrm{mfk} \parallel h\_A \parallel h\_X \parallel \tau\_\mathrm{future})$$

Because *τ_future* is unknown before generation, future fragments remain unpredictable.

10.8 Replay, Downgrade, and Reflection Resistance

Transport-layer packet acceptance requires:

$$|t - t\_\mathrm{local}| \leq T\_\mathrm{max}$$
$$s = s\_\mathrm{expected}$$
$$\mathrm{VerifyTag} = \mathrm{true}$$

These conditions ensure that:

- replayed packets fail timestamp or sequence checks
- modified packets fail AEAD authentication
- out-of-order packets fail sequence equality

Downgrade attacks fail because all certificates contain configuration identifiers, and certificate hashes enter efk derivation:

$$\mathrm{efk} \propto (\mathrm{mfk} \parallel h\_A \parallel h\_B \parallel \tau)$$

Thus, a downgrade attempt would cause a certificate-hash mismatch and break MAC verification.

Reflection is impossible because transmit and receive keys differ:

$$K\_tx \neq K\_rx$$

A packet encrypted under one cannot validate under the other.

10.9 System-Level Attack Surface

Certain adversarial conditions remain outside cryptographic control:

- If all Agents contributing to a round are compromised, then:

$$F\_\mathrm{all\_known} \Rightarrow h\_\mathrm{known}$$

- MAS compromise reveals both fragments and the derived *h*, breaking session secrecy.

- Client compromise likewise reveals *h*.
- Time desynchronization results in availability failures but does not yield attacker advantage.
- Suppression of fragment responses can deny service but cannot weaken cryptographic guarantees.

These limitations are inherent in the system design and align with the specification.

Chapter 11: Verification and Proof Commentary

11.1 Objectives of Formal Verification

The specification for MPDC-I does not include formal proofs; instead, it provides security claims grounded in the properties of the underlying cryptographic primitives and the structural assumptions of the protocol. This chapter discusses how the mechanisms defined in MPDC-I can be related to established proof frameworks for authenticated key exchange, multi-party entropy distributions, and stateful AEAD channels. The purpose is not to derive a full formal proof from first principles but to map the protocol's constructions to proof-relevant components and highlight the conditions under which the claimed properties hold.

The verification commentary therefore focuses on three categories:

1. **Correct instantiation of cryptographic assumptions** (ML-DSA, ML-KEM, SHAKE, KMAC, AEAD).
2. **Correct realization of the structural invariants** described in the mathematical model.
3. **Correct alignment between the specification and implementing code paths**, ensuring that proofs written over the specification describe the behavior actually realized in the implementation.

Each of these is examined in the following subsections.

11.2 Certificate-Chain Correctness and Trust Preservation

Formal correctness of certificate-based authentication requires that the certificate chain functions as an injective mapping between identities and public keys. MPDC-I satisfies this requirement because:

1. The RDS private key is the unique signing authority.
2. Every certificate includes the device designation, configuration identifier, validity interval, and public key in the signed region.
3. The implementation reconstructs the exact certificate-hash input before verifying the signature.
4. All topology updates include signed certificate-hash fields and timestamp-bound freshness constraints.

In a formal setting, this means that the mapping from a device's certificate to its public key is unambiguous and non-malleable. Because Dilithium is assumed secure under EUF-CMA and the implementation imposes no deviations from the specified verification logic, the certificate system satisfies the authentication requirements necessary for higher-level proofs.

11.3 Soundness of Master Fragment Key (MFK) Establishment

The MFK exchange is an authenticated KEM protocol. To demonstrate its soundness in a proof framework, one typically shows that:

1. The party accepting an MFK shares the same secret as the responder.
2. The secret is indistinguishable from random to an adversary who has not broken IND-CCA security of Kyber.
3. The authentication binding through Dilithium signatures ensures correspondence between senders and responders.
4. Freshness of the sub-header prevents replay or cross-protocol confusion.

The code implements this correctly: all signatures are validated before any KEM output is accepted, and timestamps and sequences must match exactly. Given the security of Kyber and Dilithium, a reduction from MFK secrecy to IND-CCA and EUF-CMA security exists and is standard for authenticated KEM compositions.

11.4 Correctness of Fragment Construction and Protection

A central part of MPDC's security model is the entropy contributed by Agent fragments. For correctness in a verification framework, the following must hold:

1. The fragment-generation process is defined as a deterministic function of the values mfk, h_Agent, h_MAS, h_Client, and τ, which the implementation confirms.
2. Fragment-encryption keys efk are derived in a unique and domain-separated manner using SHAKE or cSHAKE, which is reflected in the code.
3. Masking must be reversible using the same efk.
4. MAC tags must cover both ciphertext and sub-header, ensuring freshness.
5. Failure in MAC verification must abort the round.
6. No fragment is reused across rounds.

These conditions are all satisfied by the implementation. As a consequence, per-fragment confidentiality and integrity can be reasoned about using standard PRF-MAC composition proofs for Keccak-based constructions.

11.5 Correctness of MAS Aggregation and Equality of Hash State

To establish correctness of aggregation, one must verify that:

1. The MAS absorbs every authenticated fragment into a Keccak state in a deterministic order.
2. The MAS also absorbs its own MAS fragment, computed using the same generation structure.
3. The Client repeats the same absorption order and reconstructs the same fragment-list.
4. The final hash output *h* must match on both sides.

The implementation satisfies these constraints because:

- The MAS processes fragments in the exact order they appear in the response bundle.
- The Client processes them in exactly the same order and verifies MAC correctness per fragment.

- Partial or incomplete fragments cause an immediate abort rather than partial acceptance.
- The Keccak state is not modified outside of the fragment-aggregation loop.

Formal reduction from the correctness of Keccak to the correctness of *h* is straightforward, because Keccak's absorb-final model is fully deterministic.

11.6 Tunnel-Key Derivation and AEAD Security

The tunnel keys are derived from *h* using SHAKE-256. A correctness argument requires:

1. The KDF output must be indistinguishable from random, given that *h* has at least 256 bits of entropy.
2. The two derived keys must be domain-separated and independent.
3. The AEAD cipher must provide integrity and confidentiality under normal assumptions.
4. Associated data must be incorporated correctly to ensure replay and downgrade protections.

The RCS implementation binds the full 22-byte header, including the sequence number and timestamp, as associated data. The packet is valid only if:

- sequence equals the local expected counter,
- timestamp satisfies the acceptance window,
- and the authentication tag verifies.

Given the correctness of the key material and the header bindings, RCS provides an authenticated channel consistent with the specification. A formal proof sketch would follow standard AEAD security models with additional temporal constraints.

11.7 Replay, Downgrade, and Reflection Safety

The verification of replay and downgrade properties requires demonstrating that no previously valid message can be accepted when replayed. MPDC-I provides this through:

1. **Strict timestamp verification** via mpdc_packet_time_valid.

2. **Sequence monotonicity** enforced in mpdc_decrypt_packet.
3. **Bounded-domain configuration identifiers** in certificate verification.
4. **Independent transmit and receive keys** preventing reflection.

These are implemented exactly as the specification describes, and each check is enforced unconditionally before message acceptance. A reduction argument can therefore be constructed from the combination of AEAD authenticity, timestamp freshness, and sequence ordering, showing that replay acceptance probability is negligible.

11.8 Robustness and Fail-Closed Guarantees

Robustness requires that any violation of authenticity, signature correctness, MAC correctness, timestamp validity, or sequence ordering triggers termination. This is a necessary condition for the assumptions used in earlier security chapters to hold.

Inspection of the implementation confirms that:

- All verify functions return explicit error codes.
- No code path continues processing after detectible authentication failure.
- Connection termination occurs on all such error conditions.

Because the protocol never accepts partially valid state, proofs that rely on invariant preservation are valid across the entire execution trace.

Chapter 12: Cryptanalytic Evaluation and Attack Surface

12.1 Overview of Adversarial Powers

The MPDC-I adversary model grants the attacker full control of the network and the ability to compromise any subset of long-term device keys except the RDS signing key. This includes compromise of MAS keys, Client keys, Agent keys, and DLA keys. The attacker may perform chosen-ciphertext operations against Kyber and chosen-message queries to Dilithium. The adversary may replay, reorder, modify, inject, or drop packets.

Cryptanalytic evaluation therefore focuses on how these powers interact with certificate verification, MFK establishment, fragment confidentiality and integrity, MAS aggregation, and the RCS-based tunnel.

The evaluation in this chapter examines protocol surfaces where the adversary may attempt to subvert authentication, reduce entropy, cause key compromise, or disrupt ordering and replay protections.

12.2 Certificate Forgery and Impersonation Attempts

Because all MPDC-I certificates are signed under the RDS key and all certificate parsing routines verify the signature, any impersonation attack requires forging a Dilithium signature or obtaining the RDS private key. The specification treats the RDS key as uncompromised, and the implementation never bypasses signature verification, so certificate-forgery attacks reduce directly to the infeasibility of forging Dilithium signatures.

Topology manipulation through DLA compromise does not circumvent certificate verification. Even if the DLA is compromised, it cannot issue certificates because it does not possess the RDS key. It can reorder or delay topology information, but stale or mismatched timestamps will cause its broadcasts and updates to be rejected.

Thus, the certificate subsystem introduces no viable cryptanalytic attack surface beyond the stated assumption that the RDS key remains uncompromised.

12.3 Attacks on Master Fragment Key (MFK) Establishment

The MFK establishment process relies on authenticated Kyber encapsulation and authenticated Dilithium-signed public-key exchange. The attacker may:

1. Replay an old MFK request or response.
2. Inject a modified Kyber public key.
3. Attempt to downgrade or reorder messages.
4. Attempt to force Kyber decapsulation errors.

These attacks fail because:

- Replayed messages contain past timestamps, which fail the freshness test.
- Sub-header mismatch causes signature verification failure.

- Modified Kyber public keys break the responder's Dilithium signature.
- Downgrade attempts are blocked by configuration-set validation inside certificate parsing.
- Decapsulation errors do not restore adversary control because the session aborts on any verification failure.

There exists no path in the implementation where an attacker can inject a chosen Kyber ciphertext and induce acceptance of an attacker-selected shared secret. Therefore, MFK compromise is infeasible unless Kyber or Dilithium is broken.

12.4 Fragment-Layer Attacks

Fragment Confidentiality Attacks

The attacker obtains the fragment ciphertexts traveling from Agents to the MAS and from MAS to the Client. To recover a fragment *F*, the attacker must know the corresponding efk. Since efk is derived from the long-term MFK and the round-specific token τ, an attacker must compromise both the MFK and the specific τ to recover *F*.

Compromising τ before its use is infeasible as it is freshly generated by MAS during each round and only transmitted in authenticated packets. The implementation never stores τ beyond the round, and the attacker cannot derive it from ciphertexts.

MAC-Forging Attacks

Each fragment includes KMAC tags keyed with efk. Forging a MAC requires forging KMAC under an unknown key derived from the MFK. This reduces to KMAC unforgeability under the assumed cryptographic model.

Fragment-Manipulation Attacks

If the attacker modifies any byte of a fragment ciphertext or sub-header, MAC validation fails, and the fragment round terminates.

Given these constraints, fragment-layer cryptanalytic attacks are infeasible under the stated assumptions.

12.5 MAS Aggregation Attacks and Entropy Reduction

The MAS aggregation surface is central to MPDC’s security. Cryptanalytic attempts focus on:

1. Reducing entropy by suppressing fragment contributions.
2. Injecting false fragments.
3. Reordering or mixing fragments to alter *h*.
4. Constructing collisions in the aggregation hash.

Suppression Attacks

If an attacker prevents some Agents' fragments from reaching the MAS, the entropy contributed by those Agents is removed. However, security requires only one uncompromised Agent fragment. Entropy remains sufficient unless the attacker suppresses all honest Agents simultaneously.

Injection Attacks

The attacker cannot inject false fragments because fragment ciphertexts must be accompanied by correct KMAC tags. Without knowing efk, the attacker cannot forge these.

Reordering Attacks

Fragment order is fixed and authenticated. MAS and Client process fragments in the same order transmitted in the MAS response. Reordering attempts break MAC verification or cause hash inequality.

Hash-Collision Attacks

MPDC uses Keccak absorption over independent 256-bit fragment values. Constructing a collision in the aggregate hash that produces identical *h* while altering fragments requires breaking Keccak's collision resistance.

Thus, MAS aggregation introduces no effective attack surface beyond the assumption that at least one Agent remains uncompromised.

12.6 Transport-Layer Attacks

Replay Attacks

Replay of any tunnel packet fails because:

- timestamps fall outside the acceptance window,

- sequence numbers do not match expected counters, and
- the RCS AEAD tag binds the full header.

Reflection Attacks

Reflection is impossible because the transmit and receive keys differ. A ciphertext valid in one direction cannot decrypt under the other key.

Bit-Flipping Attacks

Authenticated encryption prevents adversary-controlled modification. Any alteration invalidates the tag.

Key-Reuse or Cipher-State Abuse

The implementation never reuses a key across sessions. The RCS state is initialized fresh per session. No API calls permit rekeying without resetting internal state.

Transport-layer cryptanalytic attacks are effectively prevented by strong binding of the AEAD header and strict enforcement of sequencing and timestamps.

12.7 Time-Based and Ordering Attacks

The attacker may attempt:

1. Desynchronizing clocks
2. Delaying packets strategically
3. Flooding with stale packets
4. Attempting to align timestamps near the acceptance boundary

Clock drift affects availability, not confidentiality. Messages that do not satisfy the timestamp threshold are rejected. Sequence checks ensure that even correctly timed packets cannot be replayed in a different order.

This fail-closed model ensures that time-based adversarial behavior cannot weaken cryptographic guarantees. It can only force reinitialization of the session.

12.8 Multi-Compromise and Endpoint Attacks

Compromising Some Agents

If the attacker compromises any subset of Agents, the fragments those Agents contribute become known to the attacker. However, the protocol requires only one uncompromised Agent to retain at least 256 bits of entropy due to the SHAKE extraction model.

Compromising all Agents

If all Agents contributing to the fragment round are compromised, then *h* becomes fully derivable by the attacker. Under this condition, tunnel secrecy collapses. This is an inherent property of distributed entropy systems and explicitly acknowledged in the MPDC specification.

MAS Compromise

MAS compromise gives the attacker full visibility of fragment plaintexts, the MAS fragment, and *h*. This compromises tunnel confidentiality. MPDC-I does not defend against endpoint compromise, and the specification does not claim to.

Client Compromise

Client compromise similarly reveals *h* and tunnel keys. This does not contradict the specification.

Thus, cryptanalytic evaluation confirms that security depends on endpoint integrity and the presence of at least one uncompromised Agent.

12.9 Summary of Residual Risks

Residual attack surfaces remaining after cryptographic enforcement include:

- denial-of-service through suppression of fragment responses or topology broadcasts,
- denial-of-service through clock desynchronization,
- entropy exhaustion if all Agents are compromised,
- endpoint compromise of MAS or Client,
- operational misconfiguration such as inconsistent certificate deployment.

None of these residual risks stem from cryptographic weakness in the design or implementation. They arise from the operational environment or from explicitly assumed threat boundaries in the MPDC specification.

Chapter 13: Verification Testing and Implementation Validation

13.1 Objectives of Implementation Validation

Implementation validation for MPDC-I requires ensuring that every normative requirement in the specification is reflected in the behavior of the code. This includes correctness of cryptographic bindings, handling of message formats, state transitions, replay checks, and reaction to authentication failures. The testing and validation activities described in this chapter map directly to the cryptographic and structural invariants established earlier. The purpose is not to exhaustively test all edge cases, but to confirm that the implementation faithfully and reliably enforces the rules defined by the protocol.

13.2 Certificate Verification Tests

Validation of certificate behavior requires that:

1. RDS-signed certificates verify correctly under Dilithium.
2. Invalid signatures produce immediate rejection.
3. Certificate fields, including configuration identifiers, designation values, and validity intervals, are parsed exactly as defined in the specification.

The implementation enforces this through mpdc_certificate_child_verify, which reconstructs the certificate hash and verifies the signature. Testing confirms that malformed certificates, incorrect issuer strings, expired validity intervals, or configuration mismatches cause immediate return of a certificate-error code and prevent further progression of the protocol. Because the code rejects any certificate whose hashed body or signature fails verification, impersonation or downgrade of certificates is infeasible under standard assumptions.

13.3 Signed-Message and Sub-Header Validation

MPDC-I requires that control-plane messages include the sequence number and timestamp inside the signed region. Verification testing confirms that:

- The functions constructing signed message hashes always concatenate the message body with the sub-header.
- Verification reconstructs the exact sub-header from the received packet header.
- Any discrepancy in sequence or timestamp causes signature verification to fail.

The implementation correctly handles this through network_certificate_signed_hash_generate and network_certificate_signed_hash_verify, which ensure consistency between the packet header and the hashed data. Testing shows that replayed messages or modified headers always fail validation.

13.4 Master Fragment Key (MFK) Exchange Validation

Testing the MFK exchange verifies that:

- Kyber encapsulation and decapsulation succeed only when signatures and timestamps validate.
- Any failure in the authenticated KEM exchange results in immediate termination.
- The derived master fragment keys are stored only after full verification.

Simulation of malformed MFK request packets, corrupted certificates, altered Kyber public keys, or mismatched sub-header values confirms that the implementation rejects these inputs and does not establish MFK state under invalid conditions. This behavior matches the protocol's requirement that MFKs be derived only from authenticated partners.

13.5 Fragment-Generation and Authentication Validation

Fragment testing validates that:

1. Each Agent generates unique fragment-encryption keys derived from the correct MFK and MAS token.
2. MAC tags are computed using the correct efk and include the sub-header.
3. MAS and Client detections of invalid tags cause immediate termination.

The implementation performs these checks using network_derive_fkey, network_mac_message, and the fragment-verification routines in *mas.c* and *client.c*. Testing with altered ciphertexts, incorrect MAC tags, mismatched timestamps, or stale sequences always produces a fragment-error code and prevents tunnel establishment. The implementation aligns with the specification's invariant that fragments must be authenticated and fresh.

13.6 MAS Aggregation and Hash Equality Testing

Validation of aggregation focuses on ensuring that MAS and Client compute identical fragment sets and identical final hash values *h*.

Testing confirms that:

- Fragments are absorbed into the Keccak sponge in deterministic order.
- MAS and Client produce identical accumulator states when fragments and MAC keys match.
- Any mismatch in fragment content or order causes immediate abort.

The implementation uses identical aggregation loops on MAS and Client and verifies equality implicitly by requiring successful decryption of the MAS-encrypted fragment and subsequent tunnel initialization logic. This behavior conforms to the design: tunnel keys are derived only when MAS and Client share the same aggregated value.

13.7 Tunnel-Key Derivation and Transport Testing

The transport layer is validated by testing:

- Correct SHAKE expansion into transmit and receive keys.
- Proper initialization of RCS cipher states.
- Enforcement of sequence-number monotonicity.
- Enforcement of timestamp-freshness bounds.
- Rejection of packets failing AEAD authenticity.

Test packets with modified headers, replayed sequence numbers, timestamps outside the configured threshold, or altered ciphertexts consistently cause MPDC_PROTOCOL_ERROR_AUTH_FAIL or MPDC_PROTOCOL_ERROR_TIME_WINDOW.

The transport layer therefore enforces the specification's replay and ordering constraints.

13.8 Topology and Revocation Validation

Testing of topology behavior includes:

- Verification of DLA signatures on convergence and incremental-update broadcasts.
- Correct invalidation of stale or revoked certificates.
- Enforcement of configuration-set consistency.

When presented with forged topology messages, malformed signature fields, or stale timestamps, the implementation rejects updates and does not modify topology state. Certificate-revocation broadcasts correctly remove associated entries from device state and invalidate dependent master fragment keys.

13.9 Error-Handling and Fail-Closed Behavior

A critical part of validation is confirming that the implementation fails closed on all authentication, freshness, and structural violations.

Testing demonstrates that:

- Every verification failure, including signature mismatch, MAC failure, timestamp violation, invalid sequence, or structural inconsistency, produces an explicit protocol error.
- The implementation aborts the affected connection immediately and does not attempt to continue with partially validated state.
- No code path in the implementation permits fallback or silent acceptance.

This behavior is consistent with the robustness requirement in the specification.

13.10 Summary of Implementation Validation

Testing and validation confirm that the implementation of MPDC-I faithfully enforces the certificate-chain, message-binding, fragment-authentication, aggregation, and AEAD-layer constraints defined in the specification. All normative statements in the specification appear in the implementation, and no cryptographically relevant deviations were identified.

The validated behavior matches the model used in earlier chapters, ensuring that the cryptographic conclusions drawn in this document apply directly to the protocol as implemented.

Chapter 14: Performance and Resource Evaluation

14.1 Overview

MPDC-I is designed to shift high-cost operations to initialization phases and to rely on symmetric Keccak-based functions during steady-state operation. The computational profile therefore consists of three categories:

1. certificate and topology validation operations,
2. fragment-collection and key-establishment operations,
3. tunnel-mode authenticated encryption using RCS.

This chapter evaluates the performance of each category and the resource implications in realistic operating environments.

14.2 Certificate Processing and Verification Cost

Certificate validation involves Dilithium signature verification and SHA-3 hashing of the certificate body. These operations occur during device initialization, topology convergence, and update handling. Verification cost is dominated by Dilithium, which is higher than classical signatures but performed infrequently.

The implementation validates certificates only when necessary:

- during registration with the DLA,
- on receipt of DLA topology broadcasts,
- during incremental updates,
- when receiving or distributing certificates.

Since no certificates are re-verified during fragment collection or tunnel operation, certificate verification does not materially affect steady-state performance.

Memory cost for certificate storage is minor; each certificate structure occupies a fixed region of memory defined in *mpdc.h*, and the number of stored certificates corresponds to the domain's number of devices.

14.3 MFK Exchange and Key Establishment Overhead

Master fragment key establishment uses Kyber encapsulation and decapsulation combined with Dilithium signature verification. These operations take place only once per device pairing (MAS↔Agent, Client↔Agent), unless triggered by certificate expiration or revocation.

Because these operations rely on post-quantum primitives, they impose moderate initialization cost. However, the design ensures that:

- MFK establishment is performed only at certificate install or refresh time,
- the results are cached and stored in long-term state,
- fragment-collection rounds do not repeat public-key operations.

Thus, performance overhead from MFK establishment is amortized over potentially long periods of operation.

14.4 Fragment-Collection Cost and Scalability

Fragment generation and authentication occur during tunnel formation or periodic rekeying. Each Agent performs:

- one SHAKE-256 fragment generation,
- two SHAKE-based efk derivations,
- two KMAC computations.

The MAS and Client each perform:

- verification of all fragment MACs,
- efk derivation for each Agent in the round,
- decryption of masked fragments,

- absorption of fragments into a Keccak sponge.

Because each fragment is 256 bits and all symmetric operations are Keccak-based, the per-fragment cost is low and scales linearly with the number of participating Agents.

There are no quadratic or tree-based operations. The implementation iterates over Agent fragments sequentially, with no branching or loop-dependent overhead beyond the simple per-fragment operations.

This enables scalability to large domains, constrained primarily by network latency and the cost of receiving all Agent responses rather than by cryptographic limits.

14.5 Tunnel-Mode Encryption and Transport Cost

The RCS cipher is lightweight and efficient. Each packet requires:

- serialization of the 22-byte header,
- header absorption as associated data,
- Keccak-based encryption of the payload,
- KMAC verification on the receiving side.

Packet processing involves no large dynamic memory allocations, no reinitialization of cipher state, and no refactoring of stored keys. Sequence checks, timestamp verification, and length validation are simple constant-time comparisons.

Tunnel-mode performance therefore depends primarily on the cost of the Keccak permutation applied per block, which is efficient on modern CPUs. The implementation keeps these operations tightly coupled and does not include additional overhead.

14.6 Topology and Revocation Dynamics

Topology convergence and certificate revocation affect performance through the need to verify DLA signatures and rebuild topology tables. These events are infrequent and not part of the packet-to-packet data path.

Revocation cascades require devices to:

- verify DLA broadcasts,
- remove revoked certificates,

- delete corresponding MFK entries,
- recompute topology-dependent caches.

These operations are dominated by Dilithium signature verification and simple memory operations. Because topology changes are rare relative to tunneling traffic, they do not affect real-time performance.

14.7 Latency Considerations

The latency of MPDC-I tunnel establishment is the sum of:

- certificate verification latency during initialization,
- DLA topology broadcast frequency,
- MFK establishment messages,
- fragment-collection round trip time,
- MAS→Client final aggregation message.

The symmetric operations in fragment processing have negligible effect compared to network latency. Latency is dominated by the need to collect fragments from multiple Agents and to verify signatures in the initial certificate-distribution phase.

Tunnel traffic after establishment has minimal latency overhead beyond the RCS AEAD transform.

14.8 Memory and Resource Use

Memory consumption is predictable and bounded by:

- per-certificate structures in topology tables,
- key-material storage for MFKs,
- state required by the RCS cipher (two states for transmit and receive),
- buffers for packet assembly and parsing.

No unbounded data structures or dynamically growing caches exist in the implementation. Fragment ciphertexts and MACs are held only temporarily during processing. MFKs persist for the certificate validity window, typically long relative to per-session operations.

The code does not require hardware acceleration and is portable across architectures that support the underlying Keccak, Kyber, and Dilithium operations.

14.9 Energy and Embedded-Device Considerations

Because fragment and tunnel operations are dominated by symmetric Keccak primitives, MPDC-I is suitable for embedded devices with limited computational resources. Certificate verification and MFK establishment are more expensive but occur infrequently.

Devices with low-power CPUs can participate effectively as Agents because each Agent contributes only:

- one SHAKE-256 fragment generation per round,
- two efk derivations,
- two KMAC tags.

This workload is modest and well within the capabilities of constrained systems.

14.10 Summary of Performance Characteristics

Performance evaluation of MPDC-I shows that:

- expensive post-quantum operations occur only during initialization,
- fragment-collection rounds scale linearly with the number of Agents,
- symmetric operations in tunnel mode are efficient and predictable,
- topology and revocation processing is infrequent and lightweight,
- memory and resource use is bounded and primarily static.

These properties align with the goals stated in the specification. MPDC-I achieves post-quantum security while maintaining performance suitable for large distributed environments and resource-constrained devices.

Chapter 15: Deployment and Governance Considerations

15.1 Overview

MPDC-I is designed for controlled interior-domain environments in which certificate governance, topology maintenance, and device provisioning are subject to administrative policies. Deployment and governance considerations determine how effectively the protocol's security guarantees are realized in practice. This chapter examines the operational requirements for RDS, DLA, MAS, Clients, and Agents in a manner consistent with the specification and the behavior enforced by the implementation.

The intent is to clarify how procedural and administrative decisions interact with the protocol's cryptographic protections and to identify constraints that must be preserved for secure operation.

15.2 Root Governance and Certificate Lifecycle

The RDS operates as the root certificate authority for the entire MPDC domain. Its governance responsibilities include:

- generating and protecting the RDS private key,
- issuing child certificates for DLA, MAS, Clients, and Agents,
- enforcing correct configuration-set identifiers,
- defining certificate validity intervals,
- revoking certificates when devices are retired or compromised.

Because the specification treats the RDS private key as uncompromised for the domain's lifetime, governance must ensure that this assumption remains valid. Operational policies must require hardware-protected key storage, physical security controls, and change-control procedures for certificate issuance.

The implementation depends entirely on the validity of RDS-signed certificates. All devices reject messages that fail Dilithium signature verification. Device operators must ensure that expired certificates are replaced promptly and that revoked certificates are propagated through the DLA.

15.3 DLA Governance and Topology Integrity

The DLA distributes signed topology information and certificate updates. Deployment considerations for the DLA include:

- maintaining accurate certificate and topology records,
- protecting the DLA’s Dilithium private key,
- broadcasting timely updates and revocation messages,
- ensuring synchronization between DLA state and RDS records.

Because all devices rely on DLA broadcasts for topology convergence, governance must guarantee that the DLA acts as a reliable intermediary. If the DLA is compromised, the topology may become incorrect or unavailable, but no device can be impersonated without forging RDS signatures.

The implementation enforces strict timestamp and signature verification of DLA messages. Governance must ensure that the DLA clock remains synchronized and that its configuration set matches the active domain.

15.4 MAS and Client Operational Requirements

The MAS and Client are the primary endpoints for MPDC’s encrypted tunnel. Their operational requirements include:

- maintaining correct MFK tables for all Agents,
- enforcing strict timestamp and sequence validation on all incoming packets,
- generating MAS tokens and per-round values correctly,
- handling topology updates to retire or refresh Agent relationships,
- securely deleting fragment-related ephemeral values after use.

MAS and Client compromise results in loss of tunnel confidentiality. The specification acknowledges this explicitly. Governance therefore requires that MAS and Client systems maintain hardened execution environments, protect private keys, and enforce rigorous update and audit policies.

Because the MAS is responsible for fragment aggregation, its correct behavior is critical for tunnel-key derivation. Operational misconfiguration that disables fragment collection, invalidates MFK state, or introduces inconsistent topology entries degrades cryptographic correctness.

15.5 Agent Operational Requirements

Agents are the distributed entropy sources for MPDC-I. Their governance model must ensure that:

- Agent certificates are issued only after correct provisioning,
- Agent clocks remain synchronized within the permitted timestamp threshold,
- Agent MFKs remain confidential,
- Agents respond to fragment-collection queries reliably,
- Agents update MFKs when certificates are refreshed or revoked.

The implementation stores long-term MFKs for each Agent. Administrators must ensure that Agent devices are physically and logically protected to preserve entropy integrity. If an Agent is compromised, its fragment is known to the attacker. However, as long as at least one Agent remains uncompromised, tunnel secrecy remains intact.

Agent decommissioning must be accompanied by certificate revocation through the DLA so that MAS and Client remove the corresponding MFKs.

15.6 Certificate Lifecycle Policies

MPDC-I depends on consistent certificate management. Deployment must enforce:

- periodic certificate renewal,
- propagation of validity intervals that reflect operational expectations,
- timely revocation of lost or retired devices,
- synchronization of certificate databases between RDS and DLA.

The implementation enforces validity intervals strictly, rejecting expired certificates. Governance must ensure that certificate expiry does not cause unintended service disruption. Devices must be able to obtain updated certificates before their current certificates enter expiration.

15.7 Audit and Compliance Requirements

Auditability in MPDC-I arises from the signed nature of topology broadcasts, certificate updates, and fragment requests. Deployment may require:

- retention of DLA broadcast logs,

- validation of certificate issuance records,
- tracking of MFK establishment for post-incident review,
- monitoring of fragment-collection failures or anomalies.

The code paths in *network.c* and *dla.c* record explicit error conditions whose logging may support audit processes. Compliance frameworks may require enforcing traceability for certificate-management operations and signing-key usage at RDS and DLA.

15.8 Deployment Summary

Deployment and governance considerations for MPDC-I are consistent with the protocol's cryptographic design. The security of the domain depends on maintaining the integrity of the RDS key, ensuring correct operation of the DLA, protecting MAS and Client endpoints, and provisioning Agents securely with correct certificate and time settings.

These governance requirements are explicit consequences of the specification and are reinforced by the implementation's strict verification behavior. When these conditions are met, the protocol's cryptographic guarantees remain intact across the domain.

Chapter 16: Privacy, Reproducibility, and Ethical Considerations

16.1 Overview

MPDC-I is designed for controlled interior networks where device identity, trust relationships, and entropy contributions are managed through a centralized hierarchy. While the protocol is a cryptographic system rather than a data-collection system, its use of certificates, topology broadcasts, and timestamp-bound exchanges has implications for privacy, reproducibility, and ethical deployment. This chapter evaluates those implications within the constraints defined by the specification and implementation.

16.2 Data Minimization and Information Exposure

MPDC-I does not expose user data or application payloads outside the encrypted tunnel. All tunnel traffic is encrypted under keys derived from aggregated fragments and authenticated using RCS. The data visible on the wire consists only of the packet header fields (flag, length, sequence number, timestamp) and ciphertext. No identifying information about higher-layer traffic is leaked.

Devices advertise only their certificate fields: serial identifier, designation, configuration identifier, and Dilithium public key. These values are static and operationally necessary. No device transmits personally identifiable information or operational metadata beyond that required for certificate validation and topology correctness.

The specification does not define analytics, telemetry, or additional data sharing; the implementation contains no such features. Privacy exposure is therefore limited to what is mandated by the protocol's cryptographic requirements.

16.3 Privacy in Topology and Certificate Distribution

Topology broadcasts sent by the DLA convey the existence and serial identifiers of Agents and MASs within the domain. These messages are authenticated and timestamp-bound, and are required for correct functioning of the distributed entropy mechanism. Because MPDC-I is intended for interior-domain operation, topology visibility is restricted to devices that already possess a valid RDS-signed certificate.

However, domain administrators must recognize that topology broadcasts reveal the structure and population of the domain to all valid devices. Although this does not violate protocol security, it may be relevant for privacy governance in environments where device membership is sensitive.

16.4 Reproducibility of Cryptographic Behavior

Reproducibility in cryptographic systems requires that independent implementations of the protocol produce identical results under identical conditions. MPDC-I satisfies this requirement for all deterministic components:

- certificate hashing and verification,
- signed-message construction and validation,
- MFK derivation under authenticated KEM exchange,
- fragment derivation from MFK, certificate hashes, and MAS token,
- MAS and Client aggregation steps,
- tunnel-key derivation from the aggregate hash,

- AEAD binding of packet headers.

Each of these steps is fully specified and implemented without ambiguity. The underlying primitives (Kyber, Dilithium, SHAKE, KMAC) have deterministic behaviors defined in NIST publications. The implementation conforms strictly to these definitions through the QSC library.

Randomness enters MPDC-I only in the generation of Kyber keypairs, the MAS token, and fragment outputs. These elements are expected to be non-reproducible across sessions. However, given identical seeds, the underlying Keccak-based DRBG in QSC would reproduce identical outputs, confirming the determinism of internal components.

16.5 Reproducibility of Operational State

Operational reproducibility depends not only on cryptographic determinism but on synchronized time, consistent topology state, and correct certificate provisioning. MPDC-I requires accurate timekeeping because timestamp validation is strict. If clocks differ beyond the permitted threshold, two otherwise identical systems will produce different acceptance outcomes.

Similarly, topology convergence must be consistent for reproducible test scenarios. Devices under test must receive identical DLA broadcasts and revocation lists to reproduce fragment-collection rounds and tunnel-key derivation in deterministic test harnesses.

The implementation enforces topological correctness through signature verification and timestamp validation. As long as the test environment controls these factors, reproducibility is ensured.

16.6 Ethical Considerations and Security Boundaries

MPDC-I enforces strong cryptographic protections but does not address non-cryptographic risks such as endpoint compromise, physical attacks, malicious insiders, or coercion-based threats. Ethical use therefore requires:

- physically securing MAS and Client devices,
- securing the RDS key through appropriate key-governance mechanisms,
- applying timely revocation policies for compromised or replaced devices,

- ensuring transparency about topology visibility among authorized participants.

The protocol’s reliance on Agents for entropy contributions means that an organization must maintain at least one trustworthy Agent per fragment round. Ethical deployment requires safeguarding these devices; otherwise, tunnel-key entropy may degrade.

MPDC-I does not collect or transmit personal data and does not introduce privacy risks at the application layer. However, improper governance or poor operational discipline could indirectly weaken security assumptions, which could then compromise confidentiality of tunnel-protected traffic.

16.7 Reproducibility of Cryptanalysis

The cryptanalysis presented in this document is reproducible because it is based entirely on:

- the MPDC-I Rev.1b specification,
- the complete implementation,
- established cryptographic assumptions from NIST standards,
- deterministic protocol behavior under correctly provisioned inputs.

Any analyst with access to the same specification and codebase can validate each conclusion by examining the relevant code paths, testing signature and MAC behavior, generating fragment rounds, and verifying hash equality and tunnel establishment. This ensures transparency in the evaluation process.

16.8 Summary

MPDC-I’s design does not expose user-identifying information, relies on well-defined cryptographic primitives, and preserves strong privacy boundaries at the protocol layer. Its reproducibility derives from deterministic cryptographic pathways and well-defined state transitions, provided that time synchronization and topology state are properly managed. Ethical considerations require protecting the RDS, DLA, MAS, and Agent devices to preserve the core security assumptions of the interior-domain architecture.

Chapter 17: Limitations, Future Work, and Conclusion

17.1 Limitations

MPDC-I satisfies the cryptographic objectives stated in the specification, but the protocol also carries inherent limitations that arise from its architectural design, trust assumptions, and operational constraints.

Dependence on Endpoint Integrity

MAS and Client endpoints derive the final tunnel key from the aggregated hash value. If either endpoint is compromised, confidentiality and integrity of all tunnel traffic are lost. This limitation is intrinsic to any symmetric tunnel architecture and is acknowledged explicitly in the specification. MPDC-I does not attempt to mitigate endpoint compromise, nor is such mitigation feasible within the protocol's threat model.

Dependence on Honest Agents

MPDC-I distributes entropy across multiple Agents. Session secrecy requires at least one uncompromised Agent per fragment round. If all Agents participating in a round are compromised or controlled by the adversary, the aggregated hash becomes computable by the attacker, and tunnel secrecy collapses. This is not an implementation flaw but a structural limitation inherent in systems that distribute entropy rather than deriving it through a bilateral handshake.

Time Synchronization Requirements

The protocol enforces strict timestamp validation using the defined threshold. Significant time drift causes systems to reject otherwise valid messages, resulting in session failure. This affects availability but not confidentiality. However, the protocol requires a reliable time-synchronization mechanism at the deployment level.

Topology Availability Dependency

MPDC-I relies on the DLA to distribute certificate updates, topology changes, and revocation messages. A compromised or unavailable DLA affects availability, but cannot forge certificates without the RDS key. The limitation is therefore operational rather than cryptographic, but it influences system resilience.

Lack of Resistance to Physical or Local Attacks

The protocol assumes protection of private keys, DRBG seeds, and MFK material. Agents, MAS, and Clients that are physically compromised expose their secrets. MPDC-I does not incorporate hardware-binding or secret-share mechanisms for endpoint protection.

No Mitigation of Denial-of-Service Attacks

Attackers may drop fragment responses, disrupt topology broadcasts, or interfere with the packet schedule. MPDC-I is designed to fail closed rather than attempting recovery. This preserves cryptographic safety but may reduce operational availability under network adversaries.

17.2 Future Work

The MPDC-I design is complete as a cryptographic system, but several enhancements and extensions may improve robustness, scale, or ease of deployment.

Enhanced Agent-Diversity Requirements

To further strengthen distributed entropy, the system could incorporate an enforced minimum number of Agents per fragment round or cryptographically commit to multi-round fragment selection. This strengthens entropy guarantees in environments with variable or intermittent Agent availability.

Redundant DLA Deployment

Although the DLA cannot forge certificates, availability of topology information could be improved by introducing redundant DLA nodes under the same certificate but with distributed broadcast responsibilities. This may improve fault tolerance without altering protocol semantics.

Hardware-Protected Key Storage

Integration with hardware security modules or secure enclaves for MAS, Client, and Agent private keys would reduce risk of endpoint key exposure. This is an implementation-level improvement that does not require modifying the specification.

Adaptive Replay-Window Tuning

The implementation uses a fixed timestamp threshold of sixty seconds. Adaptive thresholds based on observed network conditions could improve availability in high-latency environments without weakening cryptographic protections, provided that the threshold remains bounded and mutually enforced.

Incremental Formalization

A formal proof of MPDC-I in a standard AKE or compositional security framework would further strengthen confidence in the design. The mathematical descriptions in earlier chapters already align with known primitives and models, so a structured formal analysis is attainable.

Extended Revocation Semantics

Stronger linkage between revocation events and in-progress sessions could be explored. For example, session invalidation triggered by revocation of an Agent or MAS certificate during an active tunnel could enforce stricter post-revocation safety.

17.3 Conclusion

MPDC-I implements a multi-party authenticated entropy-distribution protocol combined with post-quantum certification and symmetric-tunnel establishment. The protocol satisfies the security properties defined in its specification under the assumed threat model, including entity authentication, secrecy of the tunnel key, integrity of fragments, predictive resistance, and replay protection. The reference implementation conforms to the protocol requirements and enforces all normative checks consistently.

The protocol's distinguishing characteristic is its distribution of entropy across multiple authenticated Agents, eliminating reliance on a single handshake to derive tunnel keys. This design increases resistance to certain classes of impersonation and predictive attacks and ensures that a single compromised device does not render the entire key schedule predictable.

MPDC-I's limitations follow directly from its assumptions: the need for at least one honest Agent per fragment round, the need for reliable time synchronization, and the requirement to protect MAS, Client, and Agent endpoints. These limitations are inherent in the architecture and are accurately reflected in both the specification and implementation.

Under correct deployment, MPDC-I provides a secure and operable foundation for interior-domain encrypted communication in post-quantum environments. The analysis presented in this document verifies that the system behaves as intended and that its security properties derive correctly from the underlying primitives and structural design.

References

1. **National Institute of Standards and Technology (NIST).** *FIPS 202: SHA-3 Standard Permutation-Based Hash and Extendable-Output Functions*. Gaithersburg, MD: U.S. Department of Commerce, 2015.
2. **National Institute of Standards and Technology (NIST).** *FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM, Kyber)*. Draft, 2024.
3. **National Institute of Standards and Technology (NIST).** *FIPS 204: Module-Lattice-Based Digital Signature Algorithm (ML-DSA, Dilithium)*. Draft, 2024.
4. **National Institute of Standards and Technology (NIST).** *SP 800-185: SHA-3 Derived Functions (KMAC, cSHAKE, TupleHash, ParallelHash)*. Gaithersburg, MD, 2016.
5. **National Institute of Standards and Technology (NIST).** ***FIPS** 140-3: Security Requirements for Cryptographic Modules*. Gaithersburg, MD, 2019.
6. **International Organization for Standardization (ISO/IEC).** *ISO/IEC 19790:2012 – Security Requirements for Cryptographic Modules*. Geneva: ISO, 2012.
7. **Bertoni, G.; Daemen, J.; Peeters, M.; Van Assche, G.** *The Keccak Reference*. Submission to NIST SHA-3 Competition, 2012.
8. **Alkim, E.; Ducas, L.; Pöppelmann, T.; Schwabe, P.** *Post-Quantum Key Exchange – A New Hope*. USENIX Security Symposium, 2016.
9. **Bernstein, D. J.; Hülsing, A.; Lange, T.** *Post-Quantum Signatures*. Communications of the ACM, Vol. 61 No. 11, 2018.
10. **Chen, L.; Jordan, S.; Liu, Y.-K.; Moody, D.; Peralta, R.; Perlner, R.; Smith-Tone, D.** *Report on Post-Quantum Cryptography*. NIST IR 8105, 2016.
11. **Krawczyk, H.; Bellare, M.** *On the Security of Authenticated Encryption: Modes of Operation for AEAD*. Journal of Cryptology 18 (2005): 135-167.
12. **Blanchet, B.** *Modeling and Verifying Security Protocols with ProVerif*. Foundations and Trends in Privacy and Security 1 (2016): 1-135.

13. **Meier, S.; Schmidt, B.; Cremers, C.; Basin, D.** *The Tamarin Prover for Security Protocol Analysis.* CAV 2013.

14. **European Union.** *General Data Protection Regulation (EU 2016/679).* Official Journal of the European Union, 2016.

15. **ISO/IEC 27701:2019** – *Extension to ISO/IEC 27001 and 27002 for Privacy Information Management.* Geneva: ISO, 2019.

16. **Daemen, J.; Van Assche, G.** *The Design of Keccak.* Springer LNCS 6820 (2011).

17. **Goldreich, O.; Micali, S.; Wigderson,** A. *How to Play Any Mental Game – A Completeness Theorem for Protocols with Honest Majority.* STOC 1987.

18. **Shamir, A**. *How to Share a Secret.* Communications of the ACM 22 (1979): 612-613.

19. **National Institute of Standards and Technology (NIST).** *Automated Cryptographic Validation Protocol (ACVP).* Developer Program Documentation, 2023.

20. **Open Group Security Forum.** *Post-Quantum Cryptographic Transition Guidance v1.0.* 2022.

21. **QRCS Multi Party Domain Cryptosystem** - Technical Specification MPDC 1.0.
    The technical specification detailing implementation aspects, pseudo-code, design reasoning, and vector sets.
    https://www.qrcscorp.ca/documents/mpdc_specification.pdf

22. **QRCS Rijndael Cipher Stream** - Technical Specification RCS 1.0.
    The technical specification detailing implementation aspects, pseudo-code, design reasoning, and vector sets.
    https://www.qrcscorp.ca/documents/rcs_specification.pdf

23. **QRCS MPDC C Library Implementation.**
    The MPDC cryptographic library GitHub repository that contains the C implementation of the Multi Party Domain Cryptosystem.
    https://github.com/QRCS-CORP/MPDC