Network Working Group
Internet-Draft
Intended status: Informational
Expires: 28 August 2026

L. Muscariello
M. Papalini
M. Sardara
S. Betts
Cisco

24 February 2026

Secure Low-Latency Interactive Messaging (SLIM)
draft-mpsb-agntcy-slim-01

Abstract

This document specifies the Secure Low-Latency Interactive Real-Time Messaging (SLIM), a protocol designed to support real-time interactive AI applications at scale. SLIM provides the transport layer for agent protocols (for example, A2A and MCP), combining gRPC over HTTP/2 and HTTP/3 with secure messaging, group communication, and native RPC semantics. The protocol provides mechanisms for connection management, stream multiplexing, and flow control while maintaining compatibility with existing gRPC deployments and supporting end-to-end encryption via MLS.

Discussion Venues

This note is to be removed before publishing as an RFC.

Source for this draft and an issue tracker can be found at https://github.com/agntcy/slim-spec.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 28 August 2026.

Table of Contents

1. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Introduction

2.1. Summary

The Secure Low-Latency Interactive Messaging (SLIM) protocol addresses the unique communication requirements of modern AI agentic applications by providing a secure, scalable, and efficient messaging infrastructure. SLIM combines the reliability and performance of gRPC with secure messaging and group communication, creating a comprehensive solution for interactive AI application communication.

At its core, SLIM consists of three primary components: data plane routing nodes that forward messages based on metadata, session-layer clients that manage secure group state and reliability, and application endpoints that publish and receive encrypted content. The protocol leverages Message Layer Security (MLS) for end-to-end encryption, ensuring that messages remain confidential even when passing through intermediate nodes or experiencing TLS termination along the communication path.

The architecture is built around a distributed network of routing nodes, each maintaining connection and subscription tables to enable efficient message routing. A control plane orchestrates the system, handling node discovery, configuration management, and administrative operations. This separation of control, session, and data planes allows for scalable deployment while keeping routing nodes lightweight.

SLIM employs a hierarchical naming system based on Decentralized Identifiers (DIDs) to ensure globally unique, secure, and routable names. The name structure follows a organization/namespace/service/instance pattern, supporting anycast and unicast routing as well as service discovery. This naming scheme supports both decentralized and federated authentication models, enabling flexible deployment across different organizational boundaries while maintaining security and interoperability.

The protocol includes a session layer that abstracts the complexity of MLS operations and messaging infrastructure from applications, providing secure point-to-point and group sessions plus native Remote Procedure Call (RPC) semantics via SRPC (SLIM RPC). SRPC enables request/response and streaming RPC directly over SLIM’s secure messaging fabric (see [SRPC]), while handling authentication, encryption, connection management, and fault recovery automatically. This design enables developers to focus on application logic rather than the underlying messaging complexities.

Security is fundamental to SLIM’s design, with authentication and authorization handled through MLS groups, cryptographic client identities, and configurable access policies. The protocol supports deployment in various environments, from data center workloads and mobile applications, while maintaining consistent security guarantees and low-latency performance characteristics.

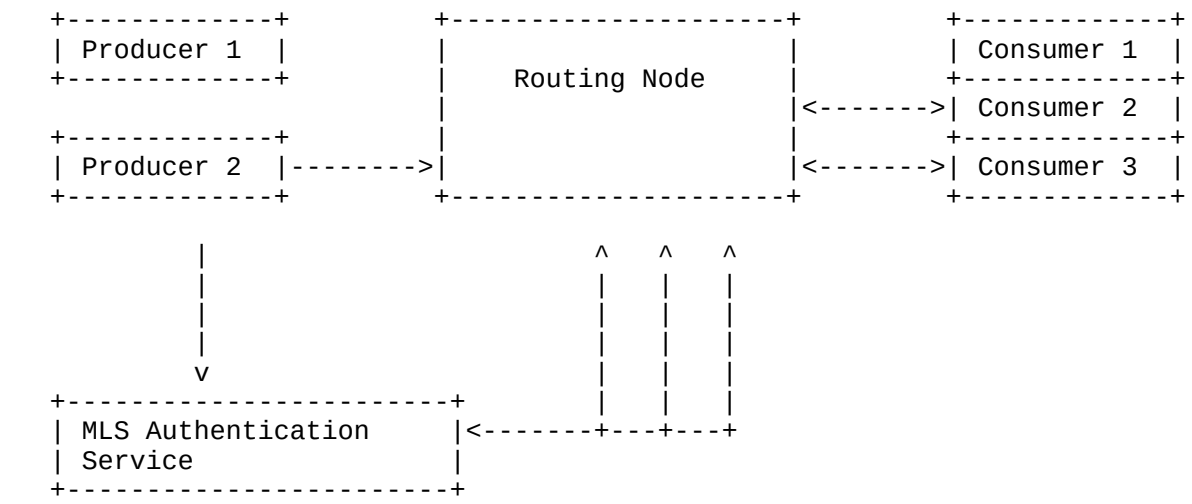
3. Protocol Overview

SLIM is designed to work as a messaging layer for applications running as workloads in a data center, but also running in a browser or mobile device while guaranteeing end-to-end security and low-latency communication. SLIM leverages HTTP/2 and HTTP/3 end to end as a thin waist of the communication stack and avoids the need to create message transcoding along the path. By leveraging message encryption via MLS [RFC9420] [RFC9750], TLS connection termination along the path does not negatively affect confidentiality. Authentication and authorization are handled at the application level and can be managed in a decentralized or federated way or a mix of both.

In SLIM there are three main communication elements: routing nodes (data plane), session-layer clients, and application endpoints that publish and receive messages.

A producer (also called a "publisher") is an endpoint that encapsulates content in SLIM messages for transport within the SLIM message network of nodes. A producer MUST belong to an MLS group to encrypt messages that can be decrypted by message consumers who are members of the same group, as specified by the MLS protocol. Once a SLIM message is encrypted, it can be published under a routable name, which is human-readable and hierarchical. This routable channel name is used by intermediate nodes to store and forward messages within the same channel, allowing consumers to retrieve messages using this name.

A routable name is a name prefix that is stored in a forwarding table (FIB). This enables requests to reach the producer and fetch a response, if one exists.



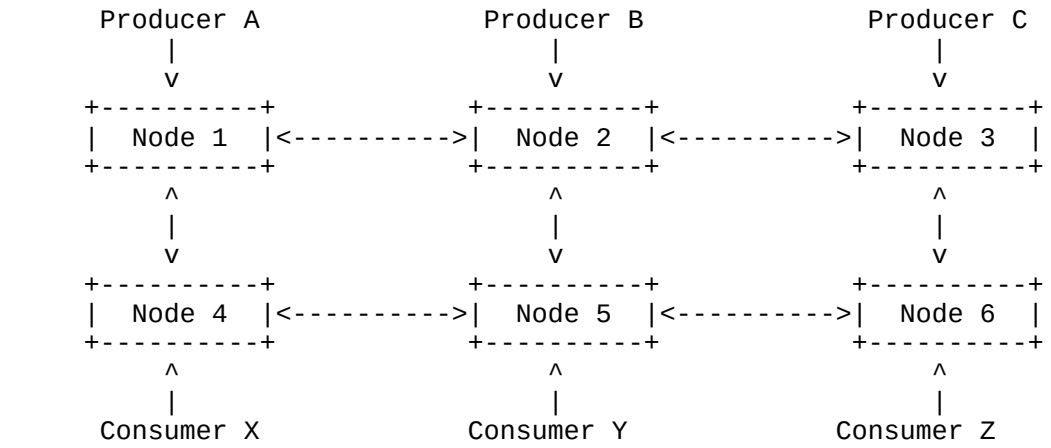
Legend:
- Producers publish to topics via routing nodes.
- Consumers subscribe to topics via routing nodes.
- MLS Authentication Service handles group authentication and key management.
- Encryption group coincides with the topic identifier.

Figure 1: Main components of the SLIM architecture.

Secure group members are clients as described in [RFC9750] which can write messages as producers or read messages as consumers. Most of the time, clients are able to read and write messages in the same secure group. Clients join secure groups as described in the MLS standard [RFC9750] via an authentication service and by exchanging messages via the delivery service. In the SLIM architecture, the SLIM nodes constitute the data-plane infrastructure that is responsible for delivering messages in a secure group via a logical SLIM channel. MLS commit messages are exchanged directly using the SLIM routing nodes.

3.1. Routing Nodes (Data Plane)

Routing nodes are fundamental components of the SLIM architecture that forward messages using metadata without inspecting payloads. They fulfill several critical functions in the messaging infrastructure. At their core, nodes efficiently route messages between connected clients while handling the distribution and delivery of messages across the network infrastructure.



Legend:
- Each Node maintains connection and subscription tables
- Bidirectional arrows represent inter-node communication paths
- Producers and Consumers connect to their local nodes
- Messages are routed through the node network based on subscriptions

Figure 2: SLIM routing node network topology.

The node architecture relies on two essential data structures that work in concert. The connection table forms the foundation for tracking all active client connections and their states, maintaining crucial metadata about each connected client. Alongside it, the subscription table manages topic subscriptions and implements message filtering rules, determining which messages should be delivered to which clients.

Through this dual-table architecture, routing nodes can effectively coordinate message delivery while maintaining optimal system performance. The connection and subscription mechanisms work together seamlessly to ensure reliable message routing, proper client tracking, and efficient subscription management across the distributed system. Each node operates autonomously while participating in the broader network, creating a resilient and scalable messaging infrastructure.

3.1.1. Connection Table

The connection table serves as a fundamental data structure within the SLIM routing node architecture, maintaining a comprehensive registry of both client-to-node and node-to-node connections. Each entry in the table contains essential metadata about connected endpoints, including their unique identifiers, connection timestamps, authentication status, and current state information.

For client connections, the table tracks end-user applications that connect to receive or send messages through the system. For node connections, it maintains the network fabric topology by recording inter-node relationships and routing paths. This dual-purpose nature enables SLIM to manage both the edge connectivity with clients and the internal communication infrastructure between nodes.

Connection states are dynamically tracked and updated to reflect the real-time status of each endpoint. This includes monitoring whether clients or nodes are actively connected, temporarily disconnected, or in various intermediate states. The table maintains crucial session information such as endpoint capabilities, protocol versions, and quality of service parameters that influence message handling.

By maintaining this detailed connection state, the table enables efficient routing decisions across the entire network fabric. It provides each routing node with immediate access to both client and node status information, allowing for rapid determination of message delivery paths and handling of connection-related events. The connection table also plays a vital role in system reliability by tracking connection health and enabling quick detection of disconnections or network issues at both the client and node levels.

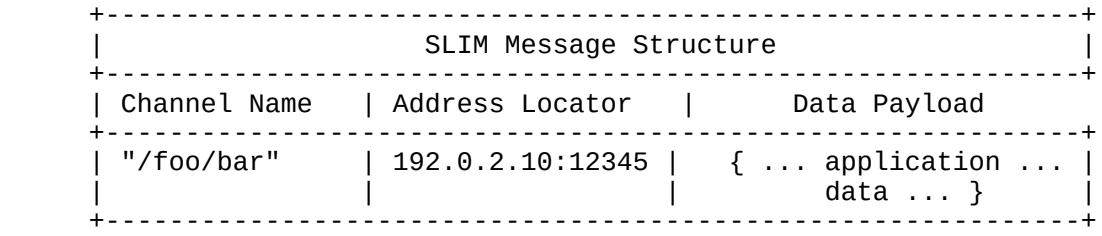
A connection table maps location-independent channel names to connections to remote nodes. The mapping is used to forward messages towards nodes that can either route messages or consume them in case consumers are directly connected to the node.

Channel names are encoded as human-readable hierarchical names for efficient table lookup operations.

3.1.2. Subscription Table and Matching

The subscription table is used to map channel subscriptions to neighboring nodes. It manages the distribution of messages based on subscriptions and ensures efficient delivery of messages. A message carries the data to be delivered as well as the channel name and the address locator of the message producer.

The control plane manages the configuration and updates of the connection and subscription tables.



Legend:
- Channel Name: Identifies the logical channel/topic for routing.
- Address Locator: Specifies the producer’s network address.
- Data Payload: Contains the actual message content.

Figure 3: SLIM message structure carrying channel name, address locator, and data.

3.2. Control Plane

The control plane is responsible for the management and orchestration of SLIM messaging nodes and their interconnections. It handles the configuration, provisioning, and monitoring of nodes, ensuring that the messaging infrastructure operates smoothly and efficiently.

Key functions of the control plane include:

* *Node Discovery and Registration*: New messaging nodes discover each other and register their presence with the control plane. This enables the control plane to maintain an up-to-date view of the messaging infrastructure.

* *Configuration Management*: The control plane distributes configuration updates to messaging nodes, including connection and subscription table updates. This ensures consistent and correct routing behavior across the node network.

* *Monitoring and Analytics*: The control plane collects and analyzes telemetry data from messaging nodes, providing insights into system performance, message flow, and potential issues.

* *Security and Access Control*: The control plane manages security policies, authentication, and authorization of nodes and clients, ensuring a secure messaging environment.

By centralizing these management functions, the control plane enhances the overall reliability, security, and performance of the SLIM messaging infrastructure. It enables efficient scaling, dynamic reconfiguration, and proactive maintenance of the node network.

3.3. Session Layer

The session layer serves as a critical abstraction component that bridges application frameworks with the underlying SLIM messaging infrastructure. It provides a unified interface that simplifies the complexity of secure messaging while handling the details of MLS client operations and message distribution.

3.3.1. Core Responsibilities

The session layer encapsulates several key functionalities:

MLS Client Operations: The layer implements comprehensive MLS client functionality including authentication procedures, message encryption and decryption, key management, and group membership operations. It handles the complex cryptographic operations transparently from the application perspective.

Channel Management: It provides seamless channel subscription and unsubscription capabilities, managing the lifecycle of channel memberships and maintaining subscription state across connection interruptions or node failures.

Message Abstraction: The session layer abstracts message passing between applications and the SLIM message distribution network, handling message formatting, routing, and delivery confirmation while providing simple send and receive primitives to applications.

Configuration Abstraction: It eliminates the need for applications to manage complex configuration details required to connect to SLIM nodes, automatically handling node discovery, connection establishment, and subscription management.

3.3.2. API Design Principles

The session layer API is designed with the following principles:

* Simplicity: Applications interact with the messaging system through intuitive publish/subscribe operations without needing to understand the underlying MLS or routing complexities.

* Asynchronous Operations: All messaging operations are designed to be non-blocking, supporting high-performance applications.

* Framework Agnostic: The API provides language bindings and framework adapters for various application development environments, ensuring broad compatibility across different technology stacks.

* Error Handling: Comprehensive error reporting and recovery mechanisms help applications handle communication issues and authentication failures.

3.3.3. Session Management

The session layer maintains persistent session state at the client across network disconnections and node failures. It implements automatic reconnection logic, subscription recovery, and message queuing to ensure reliable message delivery even in unstable network conditions. Session persistence includes maintaining MLS group membership state, channel subscriptions, and pending message queues.

3.4. Naming Considerations

SLIM requires several types of identifiers: node names, channel names, and client locators.

Node names are used for secure onboarding and authentication. Node names do not have aggregation requirements and therefore use decentralized identifiers:

node name A: did:key(node_A)

A channel name identifies a messaging group and must be routable; that is, it must include a globally unique network prefix that can be aggregated for scalable lookups and message forwarding.

A group in SLIM is an MLS group with a moderator client responsible for adding and removing group members. The moderator is identified by a cryptographic public key as defined in MLS [RFC9750], and in SLIM, also by a decentralized identifier derived as the hash of the public key [DID-W3C].

By naming entities with hashes [RFC6920], SLIM achieves secure and globally unique naming, enabling the creation of permissionless systems where channel names and client names can be distributed across administrative boundaries. W3C DIDs are optional but can be used when hash links are employed and conform to the Named Information [RFC6920] standard, referencing the IANA registry [NI-Registry].

SLIM routable name prefixes and client names can use different DID methods which have different resolution systems such as did:web [DID-Web], did:key [DID-Key] or did:plc [DID-ATProto]. See [DID-Methods] for well-known DID methods.

The naming structure follows these patterns:

client locator: did:key(org)/namespace(org)/service/did:key(client)
channel name: did:key(org)/namespace(org)/service/did:key(moderator)

Where the moderator is the special client that has the role to create a channel, add actual application clients and remove them from the group. As mentioned above the moderator is a data plane client which is a decentralized instance of the MLS delivery service as described in [RFC9420].

The hierarchical structure is required to maximize aggregation in subscription tables, which can aggregate multiple names under name prefixes such as organization identifiers, organization namespaces, and services.

3.5. Deployment Considerations

SLIM helps the deployment of agentic AI applications where a combination of data streams, tools and LLMs are combined to create a distributed multi-agent systems that can solve problems via AI.

These applications work as SLIM clients and MAY expose a service to the secure group. The channel and client naming structure combined allows for service discovery capabilities by binding the application a specific application namespace and service name.

4. RPC in Agentic Protocols and Relationship to Messaging

Most agent-facing interfaces in use today—such as A2A and the Model Context Protocol (MCP)—are Remote Procedure Call (RPC) oriented. They expose synchronous request/response semantics for tool invocation, resource listing, and capability execution. This section clarifies how RPC relates to asynchronous messaging and how the two paradigms interoperate in agentic systems.

4.1. RPC vs. Messaging: Synchronous vs. Asynchronous

* *RPC (A2A, MCP)*: The caller issues a request and blocks or awaits a timely response. Semantics emphasize tightly scoped operations (for example, “call tool X with parameters Y”) with bounded latency and explicit error contracts.

* *Messaging (AMQP, MQTT, NATS, Kafka, SLIM)*: Decoupled producers and consumers communicate via topics, subjects, or queues. Delivery can be one-to-one, one-to-many, or many-to-many, with loose coupling, buffering, and retries. Producers are not inherently blocked by consumers.

In practice, agentic applications need both: synchronous tool invocations for interactivity and asynchronous channels for streaming output, progress, coordination, and fan-out/fan-in patterns.

4.2. Challenges of RPC over Messaging

Bridging synchronous RPC semantics with asynchronous messaging infrastructure introduces several challenges:

* *Request/Response Correlation*: Messaging systems are inherently decoupled and do not guarantee a direct response path. Implementing RPC requires correlating requests and responses, often using unique identifiers and temporary reply channels.

* *Latency and Ordering*: Messaging layers may introduce variable delivery latency and do not guarantee strict ordering, which can complicate synchronous RPC expectations.

* *Error Handling*: Messaging systems may buffer, retry, or drop messages, making it difficult to propagate errors and timeouts in a way that matches RPC contracts.

* *Streaming and Multiplexing*: Supporting streaming RPC (e.g., bidirectional or server/client streaming) over messaging requires careful management of stream lifecycles, backpressure, and multiplexing multiple logical RPCs over shared channels.

* *Security and Authorization*: Ensuring that only authorized parties can invoke or respond to RPCs, and that all messages are authenticated and encrypted, is more complex in a distributed, group-based messaging environment.

The SRPC capability in SLIM addresses these challenges by providing a native, secure, and streaming RPC abstraction directly over the SLIM messaging layer, as described in the following section.

4.3. Remote Procedure Calls over SLIM (SRPC)

SLIM natively supports Remote Procedure Calls (RPC) through its SRPC (SLIM RPC) capability, enabling efficient, secure, and flexible communication patterns for distributed and agentic applications. SRPC is designed to provide gRPC-like streaming RPC semantics directly over the SLIM transport, leveraging SLIM's secure, multiplexed, and group-oriented messaging infrastructure.

4.3.1. Overview

SRPC allows applications to define and invoke remote procedures using familiar gRPC patterns, including unary calls, client streaming, server streaming, and bidirectional streaming. This enables developers to build complex, stateful, and interactive workflows between distributed agents, services, and tools, all while benefiting from SLIM's end-to-end security and group membership features.

4.3.2. Key Features

* *Streaming RPC Support*: SRPC supports all gRPC streaming modes, allowing for flexible data exchange patterns such as real-time data feeds, interactive sessions, and collaborative workflows.

* *Multiplexed Transport*: Multiple concurrent RPC streams can be established over a single SLIM connection, reducing overhead and improving resource efficiency.

* *Secure Group Communication*: SRPC leverages SLIM's MLS-based security model, ensuring that all RPC calls and responses are encrypted and authenticated within the context of secure groups.

* *Protocol Buffers Integration*: Service definitions and message schemas are specified using Protocol Buffers, enabling strong typing and interoperability across languages and platforms.

4.3.3. Architecture

SRPC is implemented as a protocol extension and a set of client/server libraries that integrate with the SLIM session layer. Developers define service interfaces using Protocol Buffers, and code generation tools produce client and server stubs that handle message serialization, stream management, and invocation logic over SLIM channels.

The SRPC protocol manages the lifecycle of RPC streams, including initiation, message exchange, error handling, and stream termination. It ensures that all communication adheres to SLIM's security and routing policies, and that group membership and authorization are enforced for each RPC interaction.

4.3.4. Benefits for Agentic Applications

By providing native, streaming RPC capabilities, SRPC enables agentic AI applications and distributed systems to:

* Orchestrate complex, multi-step workflows across multiple agents and services.
* Exchange large or continuous data streams efficiently and securely.
* Implement interactive, real-time collaboration between distributed components.
* Simplify integration with existing gRPC-based tools and ecosystems.

SRPC makes it possible to build robust, scalable, and secure agentic applications that fully leverage the power of SLIM's messaging and group communication model.

4.3.5. SLIM Naming in SRPC

SRPC derives per-service and per-handler routing names directly from the structured SLIM naming scheme to enable efficient subscription and message dispatch without requiring application developers to manually manage channel names.

Each application participating in a secure SLIM group already has a hierarchical SLIM name composed of multiple components (organization identifier, namespace, service, and client/moderator key). SRPC appends a handler-qualified suffix to the service component to form unique, routable identifiers for RPC methods.

Service interface definitions (e.g., Protocol Buffers) describe one or more RPC handlers supporting the four common gRPC communication patterns:

* Unary → Unary
* Unary → Stream

* Stream → Unary

* Stream → Stream (bidirectional)

For every handler declared inside a service, SRPC generates a method routing token using the pattern:

{package}.{service}-{handler}

Example: Given a protobuf package example_service and service Test with handlers ExampleUnaryUnary, ExampleUnaryStream, ExampleStreamUnary, ExampleStreamStream, the generated handler tokens are:

* example_service.Test-ExampleUnaryUnary

* example_service.Test-ExampleUnaryStream

* example_service.Test-ExampleStreamUnary

* example_service.Test-ExampleStreamStream

SRPC then embeds the handler token into the second component (service segment) of the full SLIM name for routing. If the original SLIM application name components c[i] are:

c[0]/c[1]/c[2]/c[3]

The subscribed name for a specific handler becomes:

c[0]/c[1]/c[2]-example_service.Test-ExampleUnaryUnary/c[3]

This transformation yields a distinct, hierarchical, and aggregatable name per RPC method while preserving the organizational and namespace prefix structure needed for subscription table aggregation. Applications exposing SRPC services automatically subscribe to all derived handler names; the SRPC library manages these subscriptions and maps incoming messages to the corresponding generated server stub functions. Developers implement only the handler logic exactly as with conventional gRPC, without manual SLIM channel management.

Benefits of this naming approach: - Deterministic and collision-resistant per-method naming. - Preserves hierarchical aggregation for routing efficiency. - Avoids separate discovery round-trips for method endpoints. - Enables capability advertisement to reference method names directly. - Simplifies multi-service deployments— adding a new handler produces a single new routable token.

This integration of SRPC handler naming with SLIM’s hierarchical naming model ensures consistent, secure, and scalable routing semantics for RPC traffic within agentic groups.

5. References

5.1. Normative References

[DID-Methods] Group, W. C. C., "Known DID Methods in the Decentralized Identifier Ecosystem", 29 April 2025, <https://www.w3.org/TR/did-extensions-methods/>.

[DID-W3C] Group, W. C. C., "Decentralized Identifiers (DIDs) v1.0", 19 July 2022, <https://www.w3.org/TR/did-core/>.

[NI-Registry] IANA, "Named Information Hash Algorithm Registry", 1 August 2013, <https://www.iana.org/assignments/named-information/named-information.xhtml>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, DOI 10.17487/RFC6920, April 2013, <https://www.rfc-editor.org/rfc/rfc6920>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC9420] Barnes, R., Beurdouche, B., Robert, R., Millican, J., Omara, E., and K. Cohn-Gordon, "The Messaging Layer Security (MLS) Protocol", RFC 9420, DOI 10.17487/RFC9420, July 2023, <https://www.rfc-editor.org/rfc/rfc9420>.

[RFC9750] Beurdouche, B., Rescorla, E., Omara, E., Inguva, S., and A. Duric, "The Messaging Layer Security (MLS) Architecture", RFC 9750, DOI 10.17487/RFC9750, April 2025, <https://www.rfc-editor.org/rfc/rfc9750>.

5.2. Informative References

[DID-ATProto]
Community, B. P., "Decentralized Identifiers (DIDs) in the AT Protocol", n.d., <https://atproto.wiki/en/wiki/reference/identifiers/did>.

[DID-Key] Group, W. C. C., "The did:key Method v0.7: A DID Method for Static Cryptographic Keys", 26 March 2025, <https://w3c-ccg.github.io/did-method-key/>.

[DID-Web] Group, W. C. C., "The did:web Method Specification", n.d., <https://w3c-ccg.github.io/did-method-web/>.

[SRPC] AGNTCY, "SLIM RPC (SRPC) Reference", n.d., <https://github.com/agntcy/slim/blob/main/data-plane/slimrpc-compiler/README.md>.

Authors' Addresses

Luca Muscariello
Cisco
Email: lumuscar@cisco.com

Michele Papalini
Cisco
Email: micpapal@cisco.com

Mauro Sardara
Cisco
Email: msardara@cisco.com

Sam Betts
Cisco
Email: sambetts@cisco.com