Using OpenMLS for Ownly Group Encryption

NDNComm 2026

Core idea

Keep the NDN/SVS/blob pipeline; let MLS manage group state and workspace keys.

Talk roadmap

Problem → MLS mechanics → TreeKEM → NDN integration → lessons

1. Problem

Why the legacy PSK+DSK scheme breaks under joins, leaves, and compromise.

2. MLS mechanics

MLS overview

How Key Package, Commit, Welcome, epochs, and exporter keys fit together.

3. TreeKEM

How TreeKEM rotates shared state with logarithmic work.

4. Integration

How Key Package / Welcome / Commit ride over the existing SVS/blob path.

5. Lessons

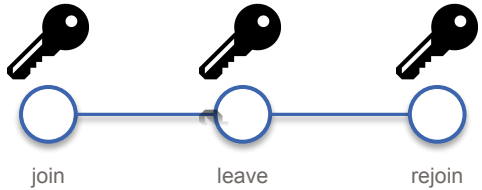
What the prototype taught us about ordering, recovery, and owner coordination.

Why the legacy static key design is not enough

Legacy Behavior

- Ownly old group encryption key: single encryption key derived from DSK + PSK
  - PSK: pre-shared key carried in join link
  - DSK: dynamic shared key fetched from existing online members after join
- Joins and leaves do not change the shared key.
- No forward secrecy or post-compromise recovery

Desired Behavior



MLS efficiently changes keys when group state changes

Core issue

Membership changes should trigger an automatic rekey

How MLS works

Message Layer Security, Standard Group Encryption Protocol, RFC9420

High-level view

- MLS organizes the group into epochs.
- Each epoch has a fresh shared secret shared by current members.
- Membership changes advance the group to the next epoch.
- Members derive the keys they need from the current epoch state.

What each member keeps

- MLS state machine
- Current epoch and transcript state.
- One key per epoch.

How one change happens

- A device publishes a proposal (keypackage)
- A committer officially changes group state and issues a Commit.
- Current members apply it; invitees process a Welcome.
- Everyone reaches the same epoch and shared key.

MLS state machine → **Exported shared key**

Why Ownly uses only part of MLS

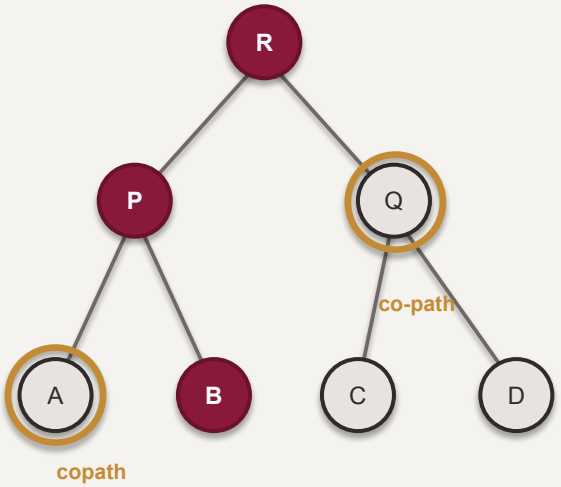
Ownly:

- Keeps NDN/SVS/blob transport
- Uses MLS for group state transitions + export shared secret for encryption

TreeKEM: scalable MLS updates

MLS is the group state machine; TreeKEM is the update mechanism inside it.

Direct path + co-path let one commit refresh the group in O(log N) work



Maroon = direct path being refreshed. Gold = the siblings used to distribute those new path secrets.

If member B updates

- B samples a fresh leaf secret and derives new path secrets: B → P → R.
- It publishes fresh public keys on that path.
- At each level, it encrypts the path secret to the copath resolution.

Why this scales

- Only O(log N) nodes on the direct path change.
- Other members recover the new root secret from the copath ciphertexts.
- No all-to-all rekey is needed.

What an MLS commit means for update, add, and remove

Update

- A member self-updates to heal from compromise or rotate secrets proactively.
- Its direct path is refreshed and everyone lands in a new epoch.

Add

- The committer incorporates a new member's Key Package.
- The new member learns the current tree through the Welcome.
- Existing members process the Commit and derive the same epoch secret.

Remove

- Removing a member is also a commit that refreshes the path to the root.
- The removed member cannot recover the new path secrets, so it loses future epochs.

Traditional group encryption: "replace one shared key somehow." MLS/TreeKEM: "advance the group state machine and derive a new epoch secret deterministically."

How OpenMLS plugs into Ownly

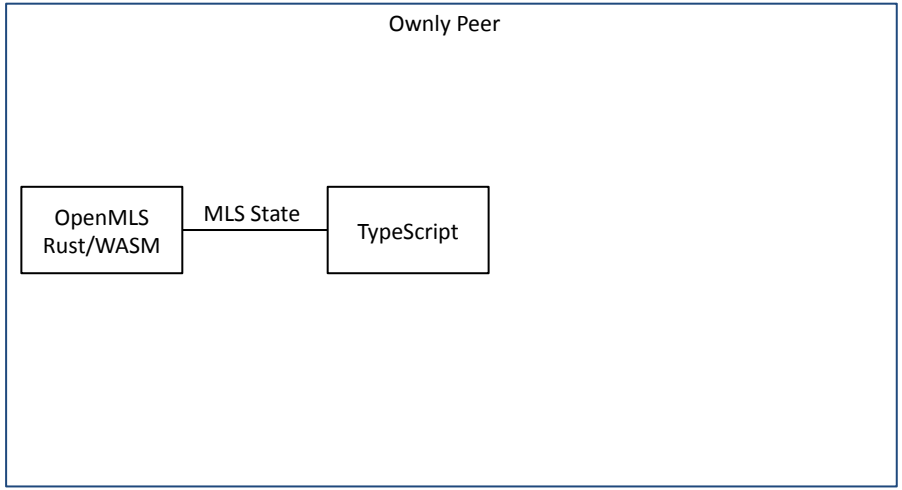
OpenMLS is the most widely used implementation of MLS in Rust

Separation of concerns

- Rust/WASM OpenMLS module:
  - MLS Local State Machine
  - Commit/Welcome processing
  - Snapshot
  - Key export.
- TypeScript orchestration
  - When to change state
  - Persisted-state handling.
- Go/WASM backend: transport of MLS messages and installation of the exported workspace key.

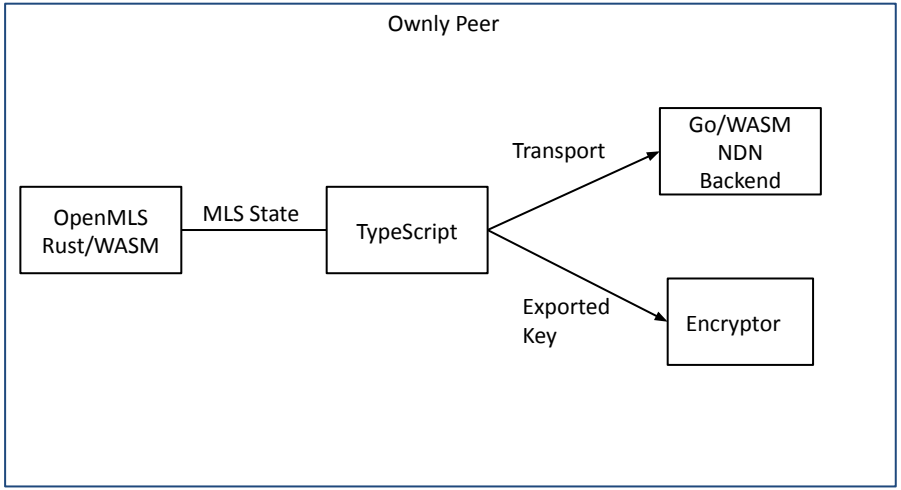
Scope today: MLS handles membership and key schedule; Ownly keeps the existing data path and encryption.

How OpenMLS plugs into Ownly



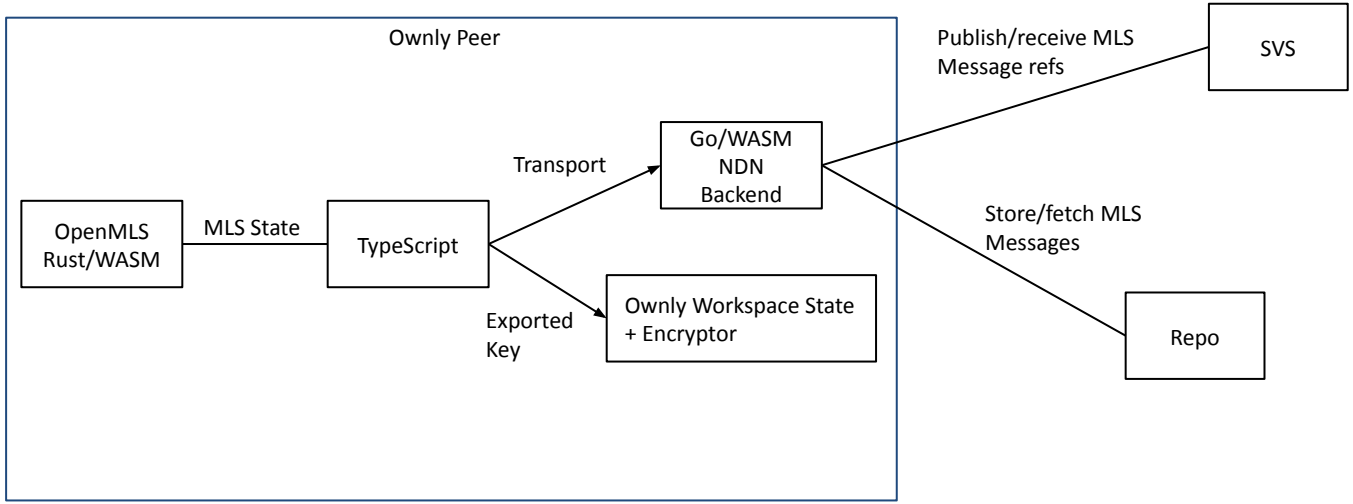
Rust/WASM holds MLS state; Go/WASM still handles SVS refs, blobs, and key installation.

How OpenMLS plugs into Ownly



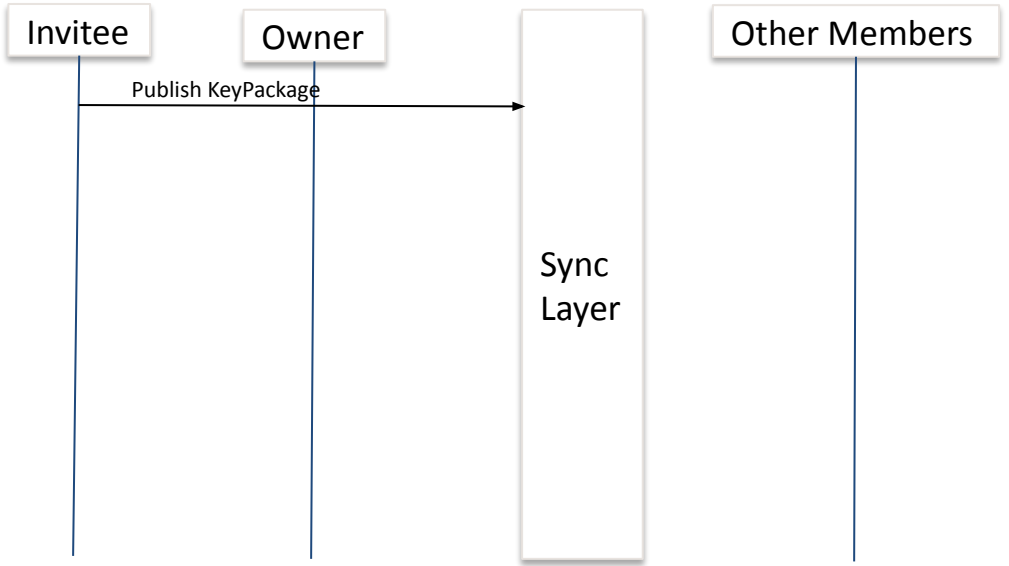
Rust/WASM holds MLS state; Go/WASM still handles SVS refs, blobs, and key installation.

How OpenMLS plugs into Ownly



Rust/WASM holds MLS state; Go/WASM still handles SVS refs, blobs, and key installation.

How a member is added in the prototype

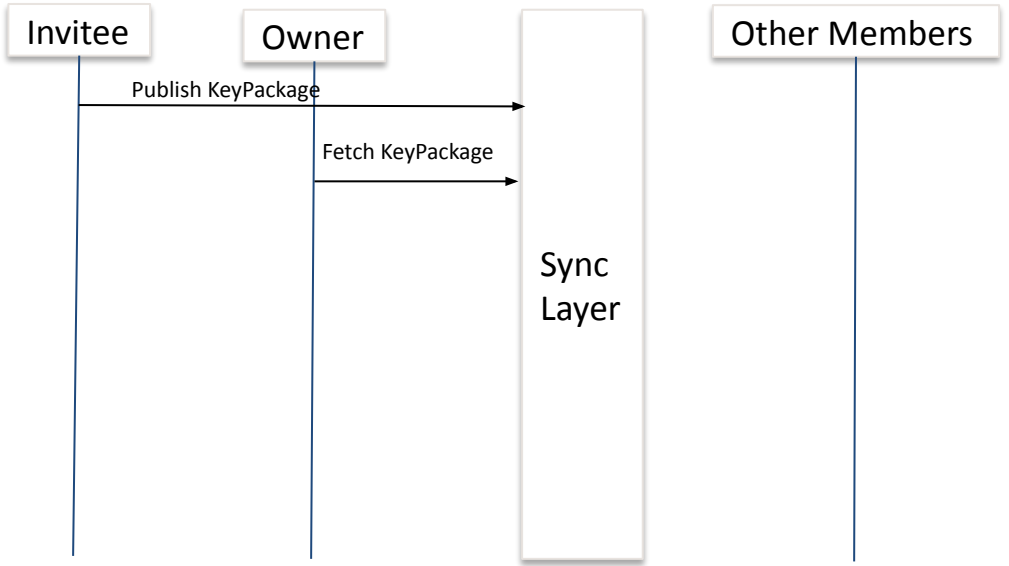


Add-member path

Each physical device is a separate MLS member

1. Invitee publishes a KeyPackage.

How a member is added in the prototype



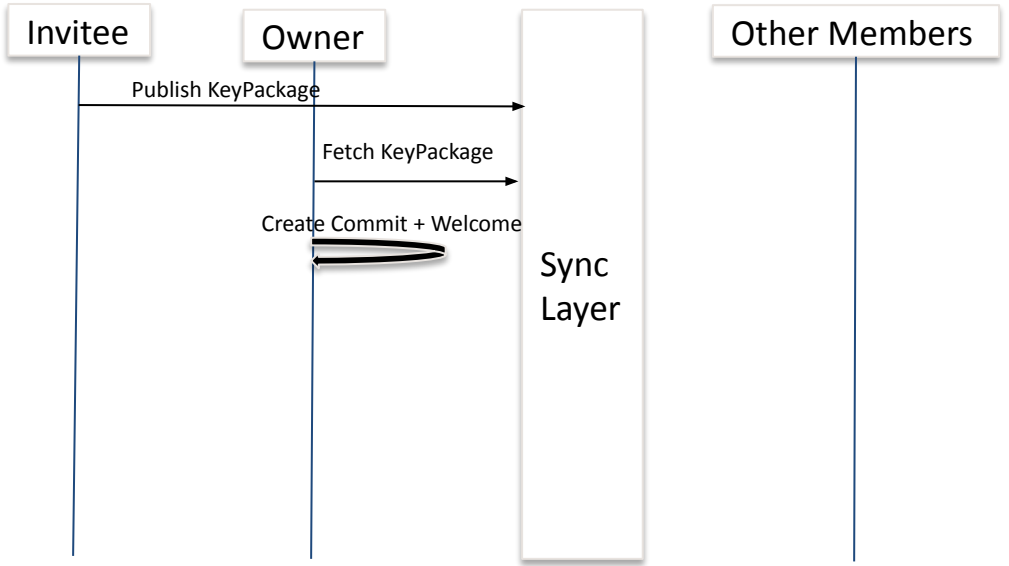
Add-member path

Each physical device is a separate MLS member

1. Invitee publishes a KeyPackage.

2. The owner fetches the KeyPackage

How a member is added in the prototype



Add-member path

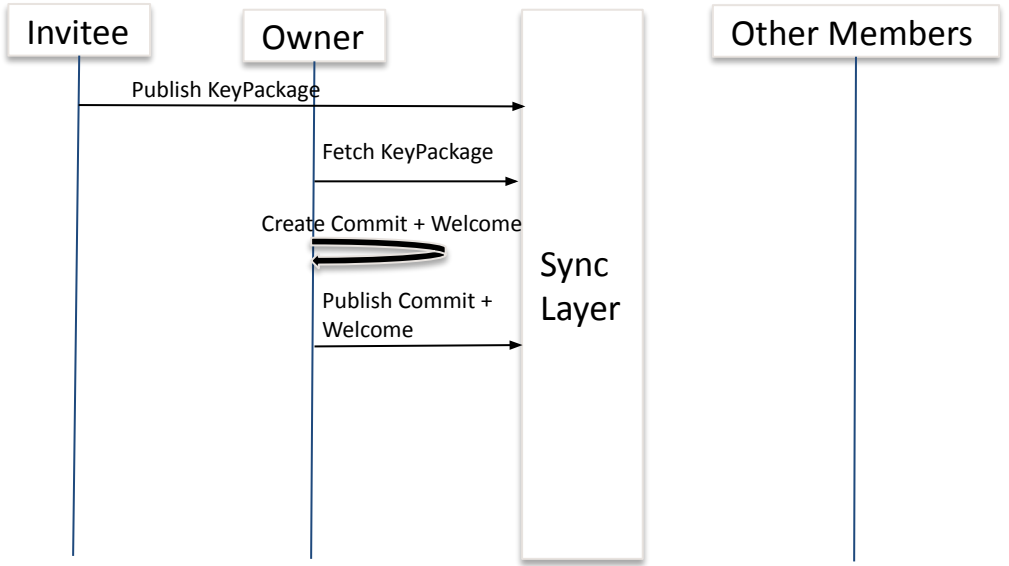
Each physical device is a separate MLS member

1. Invitee publishes a KeyPackage.

2. The owner fetches the KeyPackage

3. The owner authorizes, creates Commit and Welcome.

How a member is added in the prototype



Add-member path

Each physical device is a separate MLS member

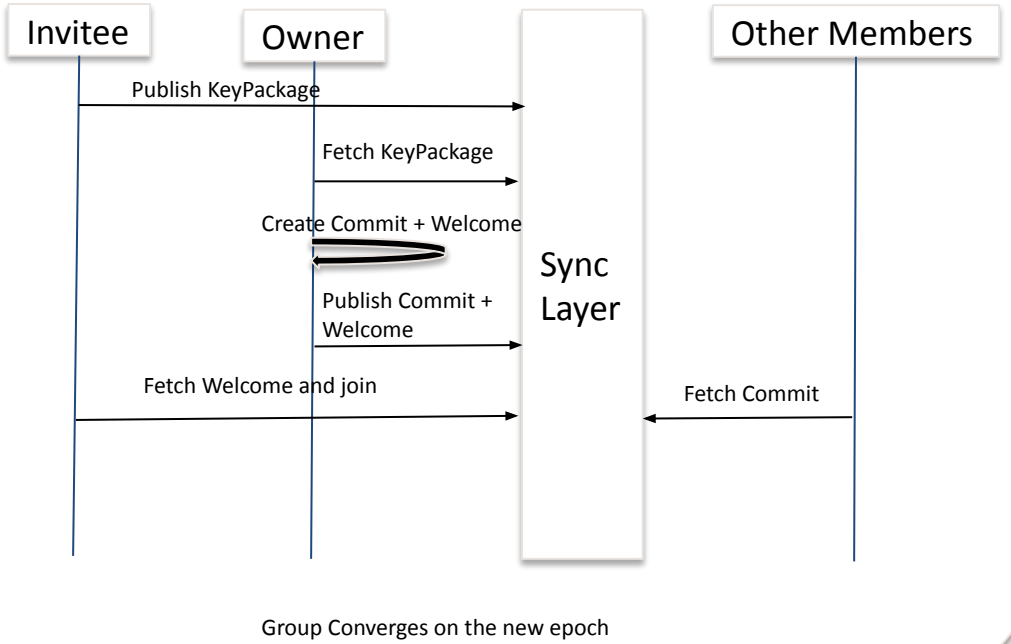
1. Invitee publishes a KeyPackage.

2. The owner fetches the KeyPackage

3. The owner authorizes, creates Commit and Welcome.

4. The owner merges commit locally, advances epoch, and publish Commit and Welcome

How a member is added in the prototype



Add-member path

Each physical device is a separate MLS member

1. Invitee publishes a KeyPackage

2. The owner fetches the KeyPackage

3. The owner authorizes, creates Commit and Welcome.

4. The owner merges commit locally, advances epoch, and publish Commit and Welcome

5. Invitee fetches the Welcome and joins, other members fetch the Commit

Persistence turns MLS into a usable collaborative system

Persist two artifacts
MLS storage snapshot
MLS group identifier

Restore on startup
Reconstruct client storage
Reload group + signing material

Catch up
Apply missed commits
Re-export active workspace key

Without persistence, every browser restart becomes a group-state failure.
Restoration must rebuild the signer, group snapshot, and active epoch schedule together.

Once MLS works, the hard part is systems behavior

Committer Conflict

If two people merge commits at the same time, it will lead to inconsistent local state and derive different keys.

Multi-device Identification

Identity is logical but each device needs its own MLS state machine.

Snapshots across key rotations

Offline members may request snapshot content created under several recent exported workspace keys, not just the latest epoch key.

Recovery after divergence

Crashes, revocations, or stale local state will inevitably occur and can leave a device's MLS snapshot inconsistent with the shared group state.

Why these issues matter

- MLS assumes a clean, consistent state machine with one committer at any given time.
- Real systems add restart, offline catch-up, batching, and partial failure.
- Requires coordination.

What the implementation needs

- Device-level identity
- Commit authority
- Snapshot key retention
- Graceful recovery after failure

Prototype policies that make it workable

Prototype rules: Use one commit sequencer, retain 5 recent keys in a key ring, and recover explicitly from divergence through MLS group state reset

Device identification

- Each device is given a MLS identity: NDN name + device ID
- Global owner-device registry selects one master owner device.
  - Only that device can update group state

Epoch-aware key handling

- A rolling key window retains several recent keys for delayed snapshot content.
  - Each key matched is matched to an epoch
- The master owner device generates a new snapshot encrypted by the new key every epoch.

Persistence + recovery

- Persist snapshot + group ID and restore on restart.
- Use explicit reset/rejoin paths when local and shared MLS state diverge.
- Tear down stale local state on revocation.

Takeaways

Cryptography result
MLS gives the missing semantics for joins, leaves, updates, and post-compromise recovery.

Systems result
The difficult engineering is ordering, persistence, catch-up, and multi-device coordination — not the crypto API itself.

Architectural result
We can retrofit MLS into Ownly without replacing the surrounding NDN application pipeline.

Key lesson:

Decentralized system still needs coordination