SafeMeet Core E2EE

Protocol Specification & Audit Readiness

Normative, implementation-grounded specification

Version 1.0

15 August 2026

Scope: SafeMeet cryptographic E2EE core only

ASSURANCE STATUS: NOT A FORMAL PROOF OR CERTIFICATION

Source freeze: SHA-256 c88928d64b07cd0789481b4f21ba9a7fe405df20dc8589158853bd75dc383210

Document control

Item	Value
Primary implementation snapshot	safe-e2ee-core-production-v1.0.zip
Snapshot SHA-256	c88928d64b07cd0789481b4f21ba9a7fe405df20dc8589158853bd75dc383210
Tested pre-release snapshot SHA-256	cc2dffc69d916ee9b1124549d37698798cc47bb16845efdb796f2c6345415d72
Tested-to-release equivalence	Dynamic fmt/check/Clippy/test/doc-test evidence was captured on the pre-release snapshot above. The final release differs in file content only by regeneration of docs/audit/AUDIT_MANIFEST.generated.json; Rust source, Cargo files, tests, and protocol vectors are byte-identical.
Repository HEAD visible in snapshot	Not included: the frozen core source archive is distributed without .git metadata.
vodozemac dependency archive	vodozemac.zip (exact sibling path dependency supplied with the release package)
vodozemac archive SHA-256	4c19093a137f5b794f56e9e4db4e1ef6b2818f41e5c751940fa23d6575c146d9
vodozemac embedded Git HEAD	4bcf7e74bf0d63efd560d86e16ed9d91dcf96d1f; archive contains local modifications, so the exact ZIP hash is the reproducibility anchor.
Branch visible in snapshot	Not available in the frozen ZIP snapshot.
Working-tree condition	Frozen v1.0 archive snapshot. Dynamic build/test evidence is supplied separately; all code-bearing files in the release archive match the tested snapshot.
Whitepaper basis	SafeMeet_Core_E2EE_Whitepaper_v1.0, 15 August 2026
Review basis	SafeMeet_PreAudit_Review_v1, informal third-party technical review
Normative language	MUST, MUST NOT, SHOULD, SHOULD NOT, MAY

Source precedence

This specification deliberately separates **what the current source implements** from **what an audit-ready deployment must additionally guarantee**. If sources disagree, the following precedence applies for factual statements about the current implementation:

1. The frozen ZIP snapshot identified by SHA-256 above.
2. Current test vectors and audit documentation inside that snapshot.
3. The v1.0 whitepaper as architecture and rationale.
4. The pre-audit review as independent critique and requested evidence.
5. This document as the normative target and audit-readiness contract.

A conflict between the normative target and the current source is not silently reconciled. It is recorded as `PARTIAL`, `HOST REQUIREMENT`, `AUDIT REQUIREMENT`, or `OPEN ISSUE` in the audit gap register.

Status legend

Status	Meaning
IMPLEMENTED	The inspected source contains the described mechanism and a directly corresponding API or test evidence.
PARTIAL	Important pieces exist, but the end-to-end guarantee is incomplete or not structurally mandatory.
HOST REQUIREMENT	The guarantee requires application, backend, OS, HSM, or deployment behavior outside this crate.
AUDIT REQUIREMENT	Evidence or independent analysis is still required even if code exists.

Status	Meaning
OPEN ISSUE	The source does not yet define or implement enough behavior to make the desired claim.
SUPERSEDED	Earlier whitepaper wording no longer matches the inspected implementation.

Contents

1. Purpose, scope, and claim discipline

1.1 Purpose

This document is the audit-facing protocol specification for the SafeMeet E2EE core. It is intended to give a cryptographic reviewer enough precision to reconstruct message formats, signed transcripts, KDF inputs, state

transitions, replay rules, persistence assumptions, and deployment dependencies without reverse-engineering the Rust implementation first.

The document also serves as an **audit readiness contract**. It identifies which guarantees are already implemented, which depend on the host application or backend, and which remain open protocol or assurance work.

1.2 In scope

The specification covers:

- device identity keys and public device bundles;
- ML-KEM public material and dual classical/PQ authentication;
- hybrid initial session establishment over an existing classical Olm context;
- one-to-one classical and sparse post-quantum ratchets;
- final hybrid message-key derivation and AEAD authentication;
- canonical framing and binary parsing;
- replay detection, reordering, skipped keys, pending epochs, and bounded state;
- transactional receive semantics and persistence contracts;
- local anti-rollback interfaces;
- device trust and verification evidence;
- sender-key room encryption and pairwise PQ-protected room-key distribution;
- verified room-state policy;
- encrypted backup/export envelopes;
- negative vectors, fuzz targets, and audit evidence requirements.

1.3 Out of scope

The core does not itself provide account authentication, a production messaging server, a real prekey directory, atomic server-side prekey claim, delivery guarantees, a production Key Transparency log, secure-hardware storage, device attestation, metadata hiding, spam/abuse prevention, push-notification privacy, a complete recovery UX, or a full post-quantum MLS/group-agreement protocol.

Compromise of an already-authorized endpoint after plaintext has been decrypted is outside the cryptographic protocol guarantee. Malware with process-memory access can read plaintext and live keys regardless of wire-protocol security.

1.4 Claim discipline

SafeMeet MUST NOT be described as “formally verified”, “cryptographically proven”, or unqualifiedly “quantum safe” on the basis of the current snapshot. The implementation contains substantial post-quantum mechanisms, but the custom composition and state machine do not yet have a complete symbolic or computational proof.

The strongest supportable claim for the current one-to-one design is that it **intends** to combine a classical X25519-derived message-key contribution and an ML-KEM-derived sparse-epoch contribution through a domain-separated HKDF-SHA256 combiner, with commit-after-authentication receive semantics. Whether the full state machine provides the desired forward-secrecy and post-compromise-security properties against all active adversarial schedules remains an audit and formal-analysis question.

Group messaging MUST be described as **sender-key group encryption with PQ-protected pairwise room-key distribution**, not as a full post-quantum group key agreement or per-message group PCS protocol.

2. Conformance model and system boundary

2.1 Production entry point

Production callers SHOULD use:

```
safe_e2ee_core::production::prelude::ProductionE2eeContext
```

Lower-level constructors and migration/test helpers MUST be treated as advanced integration APIs and MUST NOT be used to bypass identity, replay, trust, or persistence policy in production.

2.2 Core responsibilities

The E2EE core owns canonical framing, cryptographic policy, expected-identity verification, hybrid setup, AEAD operations, ratchet/replay logic, room-key authorization policy, secret-state storage contracts, Key Transparency/cross-signing verification interfaces, and backup-envelope cryptography.

2.3 Host/backend responsibilities

The host system MUST provide or integrate, as applicable:

- authenticated account/device registration;
- public device-bundle and prekey publication;
- atomic claim semantics for one-time or epoch ML-KEM public keys;
- ciphertext relay and synchronization;
- an audited implementation of `E2eeSecretStore`;
- a monotonic version source if local rollback resistance is claimed;
- Key Transparency log/auditor/monitor and verification UX if transparency resistance is claimed;
- safety-number or cross-signing verification UX;
- group rotation triggers and membership lifecycle;
- cloud backup transport/retention and recovery UX;
- target-platform crash-durability testing;
- incident response and dependency-update policy.

2.4 Server confidentiality boundary

A compatible server MAY store public device bundles, public prekeys, ciphertext messages, encrypted room-key events, room membership metadata, and sync/delivery metadata. It MUST NOT receive plaintext messages, user private identity keys, ML-KEM private keys, room chain keys, one-to-one ratchet chain secrets, or backup recovery keys.

3. Security goals and threat model

3.1 Protected assets

Asset	Security objective
Device private identity keys	Remain on authorized clients; never serialized to untrusted server state.
ML-KEM private/prekey material	Remain client-side; consumed/retired according to lifecycle policy.
Olm and EC-PQ epoch-ratchet state	Confidential and integrity protected at rest; not silently rollbackable when rollback protection is claimed.
Message plaintext	Exposed only after successful authentication and required state commit.
Room keys	Shared only with eligible, authorized current devices.
Replay state	Prevent duplicate authenticated object from causing plaintext exposure or session reinstall twice.
Trust state	Prevent silent accepted identity replacement after pinning/verification.
Backup recovery key	Never embedded in the backup envelope; required for decryption.

3.2 Adversary capabilities

The protocol assumes an attacker can observe, block, delay, reorder, duplicate, replay, truncate, mutate, and cross-context transplant network objects. The server or directory may be malicious and may return wrong, stale, or substituted public material. Inputs may be oversized or malformed to exercise parser and resource-exhaustion paths. The attacker may trigger storage/transaction failures and, unless a trusted monotonic store is deployed, may restore an older local file snapshot.

The attacker may selectively drop PQ epoch updates while delivering classical traffic. This adversarial scheduling case is particularly important because post-quantum recovery claims depend on epoch progress rather than merely on the existence of ML-KEM code.

3.3 Adversary limitations

The core alone does not defeat a fully compromised endpoint, malware that reads process memory, physical compromise without secure storage, server equivocation without transparency evidence, or disk rollback without a monotonic source.

3.4 Required fail-closed rule

For malformed input, unexpected identity, stale/invalid trust evidence, replay, tamper, unsupported version/suite, failed persistence commit, or missing mandatory PQ protection, the production path MUST return an error or an explicit buffered/pending result. It MUST NOT expose plaintext or silently advance long-lived state.

4. Trusted computing base and dependencies

4.1 Rust implementation boundary

The crate forbids unsafe Rust at the crate level. The inspected dependency profile includes:

Component	Version in inspected lockfile	Purpose
vodozemac	0.10.0, local path	Classical Olm/session primitives and identity types
pqcrypto-mlkem	0.1.1	ML-KEM-768
pqcrypto-mldsa	0.1.2	ML-DSA-44
pqcrypto-traits	0.3.5	PQ trait interfaces
x25519-dalek	2.0.1	X25519 message-ratchet branch
chacha20poly1305	0.10.1	XChaCha20-Poly1305
hkdf	0.12.4	HKDF-SHA256
sha2	0.10.9	SHA-256

4.2 vodozemac boundary

The initial hybrid setup extends an already-established classical Olm session context. Therefore, the correctness and security posture of vodozemac and the session identifier supplied by the host are part of the effective trust boundary.

AUDIT REQUIREMENT: the final audit package SHOULD record the exact vodozemac source tree used, how all-zero X25519/shared-secret cases are handled, whether any legacy Olm v1 path is reachable in production, and any locally applied patches. The current source snapshot establishes the dependency version but does not by itself establish those behavioral claims.

4.3 PQ library secret handling

The SafeMeet wrappers use zeroizing buffers for copied secret bytes and avoid secret-bearing debug output. Opaque secret-key objects are provided by upstream PQ libraries; complete zeroization of their internal allocations depends on those upstream implementations and allocator behavior. This dependency MUST be included in the constant-time/zeroization audit scope.

5. Cryptographic algorithms and parameters

5.1 Canonical cipher suite

The canonical frame layer currently recognizes one suite identifier:

Field	Value
Cipher suite ID	1
KEM	ML-KEM-768
Classical DH/session context	X25519 / Olm
AEAD	XChaCha20-Poly1305
KDF	HKDF-SHA256
Hash	SHA-256
Classical signature	Ed25519
PQ signature	ML-DSA-44

The choice pairs ML-KEM-768 with ML-DSA-44. The former targets a higher NIST security category than the latter. This is not necessarily unsafe, because authentication is not retroactively broken by harvest-now/decrypt-later in the same way confidentiality is, but the rationale and public-bundle size cost MUST be explicitly justified before external audit.

5.2 HKDF primitive

Where `hkdf_derive_32` is used, the construction is HKDF-SHA256 with caller-supplied salt, input keying material, and `info`, producing exactly 32 bytes.

`session_salt(session_id)` is SHA-256 over the raw UTF-8 session identifier.

5.3 AEAD

Message encryption uses XChaCha20-Poly1305 with a fresh random 24-byte nonce and a 32-byte key. Authentication failure MUST be indistinguishable at the protocol API from other cryptographic invalidity and MUST NOT commit staged ratchet state.

6. Encoding rules and canonical MPQFRAME

6.1 Primitive encoding

All fixed-width integers are little-endian. Variable-length byte strings and UTF-8 strings use a 32-bit little-endian length prefix:

```
LP(x) = u32le(len(x)) || x
```

`encode_info(fields)` is the concatenation of `LP(field_i)` for all fields in order. Parsers MUST reject truncation, arithmetic overflow, invalid UTF-8 where a string is required, internal length mismatch, and trailing bytes after the expected object.

6.2 Canonical frame

The common frame format is:

```
magic[8]            = ASCII "MPQFRAME"
protocol_version    = u16le
cipher_suite        = u16le
frame_type          = u16le
reserved            = u16le, MUST be 0
payload             = LP(bytes)
```

Production parsing MUST reject bad magic, unknown version, unknown suite, unknown frame type, non-zero reserved bits, truncated length, oversized/inconsistent payload, and any trailing bytes.

6.3 Protocol versions

`ProtocolVersion::V1` is encoded as `1`; `V2` as `2`. Version is part of the framing contract and MUST be checked against the expected message type. A parser MUST NOT reinterpret an unsupported version as an older structure.

7. Device identity and signed public-key hierarchy

7.1 Identity model

Each device bundle binds a user identity and device identifier to classical and post-quantum public material. Verification MUST be performed against an `ExpectedDeviceIdentity` obtained from routing/device-list context; a downloaded bundle MUST NOT be trusted merely because its own `user_id` and `device_id` fields are internally consistent.

7.2 SignedPQPublicKey

The nested signed PQ public key has structural payload version `2` and ML-KEM-768 algorithm identifier `1`.

Signing domain:

```
matrix-vodozemac-dual-signed-pq-public-key-mlkem768-v2
```

The signing transcript is the byte concatenation, in order, of:

```
raw signing_domain
u8 algorithm_id
LP(device_id)
raw ed25519_public_key[32]
```

```
LP(mldsa_public_key)
LP(mlkem_public_key)
```

The identical transcript is signed with both Ed25519 and ML-DSA-44. Verification MUST require both signatures to pass.

Payload layout:

```
u8  structural_version = 2
u8  algorithm_id = 1
LP   device_id
32B  signer_ed25519_public_key
LP   signer_mldsa_public_key
LP   mlkem_public_key
64B  ed25519_signature
LP   mldsa_signature
```

The canonical outer frame is MPQFRAME V2, frame type `SignedPQPublicKey` (`3`), cipher suite `1`.

7.3 DevicePublicBundle

Device bundle signing domain:

```
matrix-vodozemac-dual-signed-device-public-bundle-mlkem768-v2
```

The outer signed transcript is:

```
raw DEVICE_BUNDLE_SIGNING_DOMAIN
LP(user_id)
LP(device_id)
LP(curve25519_identity_public_base64)
LP(ed25519_identity_public_base64)
LP(mldsa_identity_public_key)
LP(curve25519_one_time_key_base64)
LP(embedded_signed_pq_key_payload)
```

The bundle is signed with the same Ed25519 and ML-DSA device identity keys. The embedded `SignedPQPublicKey` is then verified against the same expected device and identity key material. The canonical outer frame is MPQFRAME V2, frame type `DevicePublicBundle` (`4`).

7.4 Verification order

A production verifier MUST:

1. Parse the canonical frame and require the expected suite/type/version.
2. Compare bundle user/device identifiers to caller-supplied expected identity.
3. Verify outer Ed25519 and ML-DSA signatures.
4. Parse and verify the nested signed PQ key.
5. Require nested device ID and signing keys to match the outer device identity.
6. Apply trust-store policy before treating the bundle as eligible for messaging or room-key receipt.

7.5 First-contact limitation

An internally valid, fully substituted first-contact bundle can still pass TOFU if the malicious directory controls the first observation. Therefore high-assurance deployments MUST NOT treat TOFU as equivalent to independent identity verification. See Section 16.

8. SafeMeet Olm-ML-KEM Session Binding

8.1 Naming

The normative name is SafeMeet Olm-ML-KEM Session Binding. It is a SafeMeet-specific construction that binds verified ML-KEM-768 material and endpoint device-bundle context to an already-established authenticated Olm session. It is not Signal PQXDH, and Signal PQXDH security analyses or proofs MUST NOT be assumed to apply. Current source, API, tests, and the initial-session cryptographic namespace use SafeMeet-specific naming; historical compatibility identifiers that remain in other protocol components do not imply shared provenance or transferred proofs.

Naming and provenance rule: specifications, APIs, user-facing documentation, and audit reports MUST use the SafeMeet names. Signal PQXDH, Triple Ratchet, SPQR, PQ3, or their published analyses MAY be cited only for comparison/lineage; those analyses MUST NOT be presented as proofs of SafeMeet.

8.2 Inputs

The initiator requires:

- a classical Olm session identifier;
- verified initiator and responder `DevicePublicBundle` objects;
- the responder's verified ML-KEM-768 public key;
- caller-supplied expected identities for both endpoints;
- initial plaintext.

The responder requires the matching classical session identifier, verified endpoint bundles, the ML-KEM private key associated with the published responder public key, and the received hybrid envelope.

8.3 Transcript hash

The current source computes:

```
transcript_hash = SHA256(
    raw classical_olm_session_id ||
    initiator_bundle.to_bytes() ||
    responder_bundle.to_bytes() ||
    raw pq_ciphertext
)
```

This is a compatibility-critical detail: this specific transcript hash does not prepend a separate domain label or LP-wrap the components. The surrounding KDFs and AAD are separately domain separated. Any future change MUST use a new protocol version or explicit migration rule and new test vectors.

8.4 Hybrid session identifier

Domain:

```
safemeet-core-olm-mlkem-session-id-v2
```

The initiator encapsulates to the verified responder ML-KEM key, obtaining `pq_shared_secret` and `pq_ciphertext`.

```
session_id_key = HKDF-SHA256(
    salt = transcript_hash,
    IKM  = pq_shared_secret,
    info = LP(session_id_domain),
    L    = 32
)
```

The current display/session identifier uses the SafeMeet-specific v2 prefix:

```
"safemeet-olm-mlkem768-v2:" || lowercase_hex(session_id_key)
```

8.5 Initial payload key

Domain:

```
safemeet-core-olm-mlkem-initial-payload-key-v2
```

```
initial_key = HKDF-SHA256(
    salt = transcript_hash,
    IKM  = pq_shared_secret,
    info = LP(initial_key_domain) || LP(hybrid_session_id),
    L    = 32
)
```

8.6 Initial payload AAD

AAD domain:

```
safemeet-core-olm-mlkem-initial-payload-aad-v2
```

AAD is:

```
LP(initial_aad_domain) ||
LP(hybrid_session_id) ||
LP(classical_olm_session_id) ||
LP(initiator_bundle_payload_bytes) ||
LP(responder_bundle_payload_bytes) ||
LP(pq_ciphertext)
```

The initial plaintext is encrypted with XChaCha20-Poly1305.

8.7 HybridEnvelope

Structural payload:

```
u8 version = 1
u8 pq_algorithm = 1   // ML-KEM-768
LP pq_ciphertext
LP aead_ciphertext
```

The outer canonical frame is MPQFRAME V2, frame type HybridEnvelope (1), cipher suite 1. The HybridEnvelope structural payload remains version 1. A V1 outer HybridEnvelope belongs to the superseded pre-rename profile and MUST be rejected by the current production parser.

8.8 Receiver requirements

The receiver MUST parse the canonical envelope, verify both bundles against expected identities, decapsulate with the correct ML-KEM private key, reconstruct exactly the same transcript, derive the same session ID and payload key, authenticate the AEAD, and only then record replay state.

9. Initial-handshake replay protection

9.1 Replay identifier

Initial-envelope replay hashing domain:

```
safemeet-core-replay-initial-olm-mlkem-envelope-hash-v2
```

The replay key is scoped to owner and peer and incorporates a domain-separated hash of the authenticated HybridEnvelope. Current persisted initial-session replay keys use the initial-olm-mlkem-v2 prefix. A successful first opening MUST make a later identical authenticated object reject as replay.

9.2 Commit ordering

The production high-level opening path follows the required discipline: reject a known replay before expensive open where possible, verify/decrypt, stage replay state, commit replay state, zeroize plaintext if replay commit fails, and only then return plaintext.

IMPLEMENTED with qualification: the safe high-level production path exists. Lower-level opening APIs remain available and therefore integrators MUST use the production wrapper or provide equivalent replay commit semantics.

10. Classical EC message ratchet

10.1 State

The EC message branch maintains local receiving key material, peer receiving public material, message counters, and bounded old receiving private keys for out-of-order handling. The default retained/skipped-key bound is 1024.

10.2 Per-message send derivation

KDF domain:

```
matrix-vodozemac-general-ec-ratchet-key-v1
```

For message number `n`, the sender creates a fresh X25519 key pair and derives a DH secret against the peer receiving public key.

```
salt = SHA256(
    raw olm_session_id ||
    u64le(n) ||
    raw sender_next_public_key[32]
)

info = LP(ec_message_key_domain) || LP(sender_next_public_key)

ec_message_key = HKDF-SHA256(salt, dh_secret, info, 32)
```

The newly generated X25519 key becomes the sender's advertised receiving key for subsequent peer traffic. Old receiving keys are retained only within the configured bound.

10.3 Receive derivation

The receiver derives candidates against the newest/current and retained receiving private keys without irreversibly mutating state. Only the candidate that participates in a successfully authenticated full message is committed.

10.4 Audit qualification

The source has staged receive semantics but does not expose an equivalently complete staged EC send transaction at the top-level EC-PQ epoch ratchet boundary. This contributes to the outbound crash-atomicity gap described in Section 14.

11. Sparse post-quantum epoch ratchet

11.1 Purpose

`GeneralSparsePQState` injects ML-KEM entropy at epoch boundaries rather than running a KEM operation on every message. A per-epoch chain key yields per-message PQ message keys, reducing bandwidth and compute while periodically refreshing the PQ contribution.

11.2 Domains and bounds

```
matrix-vodozemac-general-sparse-pq-epoch-chain-key-v1
matrix-vodozemac-general-sparse-pq-message-key-v1
matrix-vodozemac-general-sparse-pq-next-chain-key-v1
```

Maximum skipped PQ messages within an epoch: `64`.

11.3 Public/private prekey representation

The current helper generates a sequence of public epoch prekeys identified by `u64 key_id` and retains corresponding `PQKeyPair` objects in a private `HashMap<u64, PQKeyPair>`.

On successful receive/epoch commit, the private prekey used for decapsulation is removed. This is a useful one-time-consumption property, but it is **not a complete prekey lifecycle specification**.

11.4 New epoch derivation

When starting a new sparse PQ epoch, the sender selects a peer public prekey and encapsulates to it. The epoch identifier is the previous epoch plus one, beginning at zero for the first epoch.

The epoch chain key is derived using HKDF-SHA256 with a session-derived salt and context containing epoch, prekey identifier, and KEM ciphertext. The implementation's exact domain is listed above; current test coverage should be treated as the compatibility oracle for serialization and derivation.

11.5 Per-message PQ key

The PQ message key is derived from the epoch chain state and binds at least the epoch identifier, epoch message index, and global message number under the dedicated message-key domain. The next epoch/message chain transition uses the dedicated next-chain domain.

11.6 Receive behavior

If a message references an unseen epoch but carries no usable epoch update, the receiver returns a pending-epoch result rather than accepting the message. An included update may be decapsulated tentatively, but the corresponding private prekey MUST NOT be removed until full AEAD authentication and state commit succeed.

11.7 Audit-ready prekey lifecycle requirement

The current source does not define a signed epoch-prekey batch with generation, expiry, replenishment, retirement, or server-side atomic claim semantics. Therefore the following is an **audit-ready deployment requirement, not a claim about the current wire format**:

- Creation: private ML-KEM epoch keys MUST be generated by a cryptographically secure RNG and stored only through protected secret storage.
- Publication: every public prekey MUST be bound to the publishing device and an unambiguous key identifier; a production design SHOULD also bind a monotonic generation/version and validity interval.
- Claim: a server MUST NOT validly hand the same one-time public prekey to multiple initiators when one-time semantics are required; claim consistency MUST be defined and tested.
- Consumption: the receiving client MUST consume a decapsulation private key only after the corresponding authenticated epoch transition is durably committed.
- Deletion/retirement: expired, superseded, or consumed private keys MUST become unavailable to ordinary decrypt paths and SHOULD be zeroized when ownership is released.
- Replenishment: the host MUST define low-watermark and publication/retry behavior without reusing key identifiers.
- Retention bound: the maximum number of simultaneously retained decapsulation keys MUST be explicit. Eager generation of many in-flight epochs can increase compromise blast radius.

OPEN ISSUE: the exact signed publication/claim protocol and in-flight decapsulation-key policy are not present in the inspected snapshot.

12. SafeMeet EC-PQ Epoch Ratchet

12.1 Composition

The one-to-one message layer combines:

1. a per-message EC ratchet key;
2. a per-message PQ key derived from the current sparse ML-KEM epoch;
3. a domain-separated HKDF-SHA256 final combiner;
4. XChaCha20-Poly1305 authenticated encryption.

The current source has replaced the earlier direct SHA-256 final combiner described in the July whitepaper with HKDF-SHA256 V2. The whitepaper's old combiner description is therefore **SUPERSEDED** for the inspected snapshot.

12.2 Limits

Limit	Default
Pending messages	128
Skipped/classical history	1024
Maximum future PQ epoch gap	8
Sparse-PQ skip within an epoch	64

12.3 Final hybrid key derivation

Domains:

```
extract-salt domain = matrix-vodozemac-general-sparse-triple-hybrid-extract-salt-hkdf-sha256-v2
expand-info domain  = matrix-vodozemac-general-sparse-triple-hybrid-expand-info-hkdf-sha256-v2
AAD domain          = matrix-vodozemac-general-sparse-triple-aad-mlkem768-v2
```

Define context `C` as:

```
LP(olm_session_id) ||
u64le(message_number) ||
u64le(epoch_id) ||
u64le(epoch_message_index) ||
LP(sender_next_ec_public_key) ||
u8(pq_update_present) ||
[ if pq_update_present:
    u64le(update_epoch_id) ||
    u64le(receiver_prekey_id) ||
    LP(pq_ciphertext)
]
```

Then:

```
S = SHA256(raw extract_salt_domain || LP(C))
IKM = LP(ec_message_key) || LP(pq_message_key)
Info = raw expand_info_domain || LP(C)
FinalKey = HKDF-SHA256(salt=S, IKM=IKM, info=Info, L=32)
```

Temporary IKM and component message-key buffers SHOULD be zeroized as soon as possible after deriving the final AEAD key.

12.4 Message AAD

The authenticated metadata contains the AAD domain, session identifier, message number, epoch identifier, epoch message index, sender next EC public key, PQ-update presence flag, and, when present, the update epoch, receiver prekey ID, and ML-KEM ciphertext. A change to any bound field MUST cause AEAD failure.

12.5 EC-PQ epoch envelope

Structural payload version is `1`, while the canonical outer protocol frame is V2 because the current EC-PQ epoch ratchet profile uses the V2 HKDF combiner.

```
u8  structural_version = 1
u64 message_number
LP   sender_next_ec_public_key
u64 epoch_id
u64 epoch_message_index
u8  pq_update_present
```

```
if present:
    u64 update_epoch_id
    u64 receiver_prekey_id
    LP  pq_ciphertext
LP   aead_ciphertext
```

Outer frame: MPQFRAME V2, frame type EcPqEpochEnvelope (numeric ID 2), cipher suite 1. The numeric frame-type identifier is unchanged by the terminology remediation. A V1 outer frame for the current EC-PQ Epoch Ratchet profile MUST be rejected.

12.6 Current known-answer vector

The current repository contains a known-answer vector in tests/test_vectors/current/ec_pq_epoch_ratchet_kdf_aad_current.json. For the recorded inputs, the expected final key is:

```
6875c28b83a36deac157135355fe0122c256717706d2b5c93c0d1e0449705ded
```

This vector SHOULD be treated as a release compatibility gate. Any intentional KDF change MUST change the protocol profile/version and publish a new vector set rather than updating the expected output in place without protocol review.

13. Message receiving, reordering, replay, and concurrency

13.1 Required receive state machine

The normative receive ordering is:

```
parse canonical frame
-> validate version/suite/type and bounds
-> replay precheck
-> prepare PQ state or return PendingEpoch
-> prepare EC candidate state
-> derive hybrid key
-> authenticate AEAD
-> stage durable replay/ratchet records
-> commit required durable transaction(s)
-> commit in-memory ratchet state
-> expose plaintext
```

An authentication failure MUST leave long-lived ratchet and replay state unchanged.

13.2 Pending epochs

A message whose required PQ epoch is not yet available MAY be buffered, subject to the pending-message limit. Buffering is not successful decryption and MUST NOT record the message as replayed. Exact duplicate pending objects MUST be rejected rather than stored repeatedly.

13.3 Reordering

Within configured bounds, skipped EC/PQ keys allow delayed messages to be decrypted after later messages. When a gap exceeds a configured bound, the receiver MUST fail or require explicit recovery rather than allocate unbounded attacker-controlled state.

13.4 Replay identifiers

Replay keys are context scoped. Current domains include:

```
safemeet-core-replay-initial-olm-mlkem-envelope-hash-v2
matrix-vodozemac-replay-triple-envelope-session-id-hash-v1
matrix-vodozemac-replay-room-key-event-hash-v1
```

The EC-PQ epoch replay key binds owner, peer, session, epoch, global message number, and epoch message index. A successfully committed authenticated EC-PQ epoch message MUST not expose plaintext a second time under the same replay identity.

13.5 Scalable replay journal

The persistent replay implementation uses immutable `.rpl` segment files, delta-oriented transactions, commit locking, checksums, atomic rename, and optional compaction. Segment limits include up to 1,000,000 entries and a 256 MiB body limit. The lock loop is bounded.

The segment checksum is an **unkeyed SHA-256 integrity check** intended to detect torn/corrupt data. It is not sufficient to authenticate storage against an attacker who can edit files and recompute checksums. Production malicious-local-writer resistance therefore requires the secret-store/rollback architecture in Section 14.

13.6 Concurrency

The replay transaction reloads disk state under the commit lock and rejects a concurrently committed duplicate. Host code MUST still serialize or coordinate concurrent ratchet writers for the same logical session. A file-level replay lock is not a general session-state linearizability mechanism.

14. Persistence, crash safety, and rollback resistance

14.1 Transaction contract

`e2ee_state_transaction.rs` defines typed records for ratchet, replay, trust, room, device-session, pending-message, and production-receive commit-marker state. Record AAD binds user, device, record type, scope, and version.

The source also contains an atomic file batch format for related state records, with temporary-file write, file sync, atomic rename, and checksum. A stronger helper can stage actual ratchet and replay bytes plus a commit-marker digest that links both records.

14.2 Important distinction: checksum versus secure storage

The atomic batch checksum is unkeyed SHA-256. It can detect corruption and incomplete writes but does not authenticate data against a malicious local writer. Comments in the code explicitly assume values are already protected by the host. Therefore the batch journal MUST NOT be presented as a replacement for an authenticated `E2eeSecretStore` or a monotonic anti-rollback mechanism.

14.3 Receive persistence status

The production transactional decrypt path delays plaintext return until its sequence of application transaction, replay commit, and in-memory ratchet commit succeeds. This is valuable, but the inspected path does not yet use one single atomic durable transaction containing the actual replay and ratchet state across all storage domains.

Consequently the intended invariant is stronger than the currently integrated implementation:

```
Desired:
AEAD success -> one atomic durable ratchet+replay commit -> expose plaintext

Current high-level integration:
AEAD success -> stage/commit host transaction markers -> commit replay journal -> commit in-memory ratchet
-> expose plaintext
```

PARTIAL / release-blocking for the strongest crash-safety claim. The existing atomic-state-batch primitive SHOULD be integrated into the real production receive path or the host transaction interface SHOULD provide a documented equivalent with fault-injection tests.

14.4 Outbound crash atomicity

A second, distinct issue exists on send. The current top-level EC-PQ epoch-ratchet send path uses EC/PQ header functions that can mutate send state before the message and durable ratchet state have been atomically published. A crash between state advancement, persistence, and network send can create skipped state or retry ambiguity.

An audit-ready outbound path MUST stage the next EC/PQ state and ciphertext, durably commit the new state with a stable send identifier, and only then expose/transmit the ciphertext according to an idempotent host policy. If the implementation instead transmits before persistence, it MUST provide an equivalent recovery protocol that prevents key/nonce/state reuse.

OPEN ISSUE (AR-16): outbound ratchet atomicity is not fully implemented in the inspected source.

14.5 Anti-rollback

`E2eeSecretStore` provides the record-level confidentiality/integrity contract; `MonotonicVersionStore` and rollback-marker helpers provide a way to compare state generation with a trusted monotonic source. The core alone cannot make ordinary local files rollback-proof.

If a deployment claims rollback resistance, the host MUST integrate a monotonic version source such as hardware-backed secure storage, an authenticated remote monotonic counter, append-only authenticated storage, or another audited mechanism. Concurrent writers MUST be serialized around version updates/CAS.

14.6 Platform durability

Directory metadata sync is performed on Unix where supported. On non-Unix/Windows, the helper is best-effort/no-op for directory fsync semantics. Therefore target-platform power-loss/crash guarantees MUST be validated with fault-injection testing rather than inferred solely from Rust `rename` behavior.

15. Multi-device and offline-recipient behavior

15.1 Device isolation

Each device has distinct identity and public bundle material. Expected identity always includes both user and device identifiers. A valid bundle for another device MUST not be accepted as a substitute.

15.2 Pairwise session fan-out

A sender establishing one-to-one messaging with multiple recipient devices MUST establish/maintain a cryptographic session for each eligible device. Group room keys are likewise sealed separately for eligible recipient devices through one-to-one channels.

15.3 Offline recipients

Offline delivery may reorder classical messages and sparse PQ epoch updates. The bounded pending/skipped-key machinery is the core mechanism for tolerating such reordering. The host SHOULD retain and redeliver required epoch-update ciphertexts until the receiver can make progress, while respecting replay rules.

OPEN ISSUE: the source does not define a complete adversarial-drop progress protocol for PQ epoch material. A malicious relay that selectively suppresses epoch updates can affect PQ recovery/progress. The audit-ready design MUST define when the sender creates an epoch, whether only one decapsulation key may be outstanding or multiple may be in flight, how updates are retransmitted, and when classical-only progress is rejected or degraded.

15.4 Prekey claim

The server-side atomic claim of ML-KEM epoch prekeys is outside the crate. A production backend MUST define claim uniqueness, authentication, stale-batch handling, replenishment, and failure recovery. Client logic MUST treat missing/invalid mandatory PQ material as an explicit error rather than silently downgrade to a classical-only session.

16. Trust, TOFU, safety numbers, cross-signing, and Key Transparency

16.1 Trust states

The trust store represents states including `Unverified`, `TofuPinned`, `Verified`, `Blocked`, `Revoked`, and `Removed`. Blocked, revoked, and removed are terminal/fail-closed for ordinary refresh; legitimate re-enrollment must be explicit rather than a side effect of receiving a new bundle.

16.2 TOFU

TOFU pins the observed Ed25519 identity and a domain-separated hash of the ML-DSA identity public key. Later replacement of the same expected device is rejected. This protects against post-first-contact silent substitution but cannot detect a malicious first observation.

16.3 Safety numbers

Safety-number verification can raise assurance above TOFU when the host exposes a reliable user ceremony. The host MUST make any “verified” UI state correspond to the exact identity material evaluated by core policy; it MUST NOT preserve a verified badge across an identity change without explicit re-verification.

16.4 Cross-signing and Key Transparency

The core contains policy/interfaces for cross-signing and Key Transparency evidence, including subject/fingerprint binding and stale/weak-evidence rejection. It does not contain a production transparency log, auditor, gossip protocol, or monitor.

High-assurance deployments SHOULD require at least one independent verification mechanism beyond TOFU before first sensitive send. A deployment MUST NOT claim server key-substitution resistance from Key Transparency unless the real backend, consistency/equivocation behavior, client verification, and UX are implemented and audited.

17. Group sender-key cryptography and room-key distribution

17.1 Sender-key model

Each outbound room sender session starts with a random 32-byte symmetric chain key. Per-message room keys and next-chain keys are derived with HKDF-SHA256 under dedicated domains, then messages are encrypted with XChaCha20-Poly1305.

Domains:

```
matrix-vodozemac-room-message-key-mlkem768-v1
matrix-vodozemac-room-next-chain-key-mlkem768-v1
matrix-vodozemac-room-message-aad-mlkem768-v1
```

Default room skipped-message bound: `1024`.

17.2 Room session identifier

The implementation derives a room session identifier from a SHA-256 hash over a room-session domain plus room ID, sender user ID, sender device ID, and the initial chain key. Because the secret chain key contributes to the identifier, a server cannot choose the same identifier without knowledge of the sender key.

17.3 Room message AAD

AAD binds room ID, session ID, sender user ID, sender device ID, and message index. Cross-room, cross-session, or cross-sender transplantation MUST fail authentication or pre-auth policy checks.

17.4 RoomKeyPayload

The room-key payload binds room, session, sender user/device, message-chain index, and the 32-byte chain key. Canonical room-key payloads use MPQFRAME V1, frame type `RoomKeyPayload` (`5`). Inbound session storage uses frame type `RoomInboundSession` (`6`), and room messages use `RoomMessageEnvelope` (`7`).

17.5 Pairwise distribution

Room keys are distributed separately to eligible devices through `OneToOneRoomKeyTransport`. Current production policy requires the transport implementation to report `is_pq_protected() == true`.

This Boolean is an integration assertion, not a cryptographic proof that a particular transport instance actually completed the SafeMeet one-to-one PQ-protected path. Audit-ready integration SHOULD replace or reinforce this assertion with a capability/type/evidence object that can only be produced by a verified PQ session establishment, or otherwise demonstrate by construction that the concrete transport cannot be classical-only.

18. Verified room state and group recovery

18.1 Authorization context

Room-key sharing policy checks active sender and recipient membership, current room epoch, current member/device epoch, sender/recipient binding, room/session binding, and trust eligibility. Revoked, blocked, removed, non-member, or stale-epoch devices MUST NOT receive current room keys.

18.2 Signed room-state chain

Verified room-state events are ML-DSA signed and form an ordered chain through previous-event identifiers. The state logic rejects duplicate event identifiers, wrong previous event, unauthorized signer role, membership/epoch inconsistency, and invalid signature.

18.3 Unsupported forwarding/request flows

Unsupported room-key request or forwarded-key flows fail closed rather than silently redistributing keys. Any future support MUST define explicit authorization, replay, provenance, and trust rules.

18.4 Post-compromise recovery qualification

The current group construction does not provide per-message post-quantum group PCS. Recovery after sender-key compromise depends on creating a fresh room session and redistributing its key to the current eligible membership.

OPEN DOCUMENTATION/PRODUCT REQUIREMENT: the deployment MUST define triggers and maximum latency for room-session rotation, including member removal, device revocation, suspected compromise, and periodic rotation. Without that policy, “group recovery” is underspecified even though the cryptographic primitive for fresh sessions exists.

19. Backup/export envelope

19.1 Purpose

The backup envelope exports typed E2EE secret records as one client-side encrypted object. Cloud/storage service behavior and recovery UX are outside this cryptographic format.

19.2 Envelope constants

```
magic               = ASCII "MPQBACKUP"
envelope version    = 1
plaintext version   = 1
schema version      = 1
salt length         = 32 bytes
nonce length        = 24 bytes
minimum recovery key length = 32 bytes
KDF domain          = safe-e2ee-core-backup-envelope-kdf-v1
AAD domain          = safe-e2ee-core-backup-envelope-aad-v1
```

19.3 Wire format

```
raw magic "MPQBACKUP"
u8  envelope_version
u16 schema_version
LP  owner_user_id
LP  owner_device_id
LP  backup_id
u64 backup_version
u64 created_at
32B salt
24B nonce
LP  ciphertext
```

19.4 Key derivation

```
backup_key = HKDF-SHA256(
    salt = header.salt,
    IKM  = recovery_key,
    info = LP(KDF_DOMAIN) ||
           LP(owner_user_id) ||
           LP(owner_device_id) ||
           LP(backup_id) ||
           u64le(backup_version) ||
           u16le(schema_version),
    L = 32
)
```

19.5 AAD

AAD binds envelope version, schema, owner user/device, backup ID, backup version, creation time, salt, and nonce under the dedicated AAD domain. Wrong owner/device context, wrong recovery key, wrong version, or ciphertext tamper MUST fail.

19.6 Plaintext record list

The encrypted plaintext contains its own magic/version and a count of typed records. Each record binds user, device, record type, record identifier, version, and secret bytes. Cross-owner/device records MUST be rejected before export/restore.

19.7 Backup limitations

The recovery key is not stored in the envelope. Password-to-recovery-key hardening, cloud-retention policy, account recovery, remote rollback resistance, and recovery phrase UX are host responsibilities.

20. Versioning, downgrade resistance, and build profiles

20.1 Production features

The crate default feature set is empty. `production` enables the selected `mlkem-pqcrypto` backend. Compile-time guards reject unsafe or incompatible production combinations including test-support, legacy migration, legacy Kyber, broad experimental PQ sets, and the alternate experimental ML-KEM RustCrypto backend.

20.2 Current frame-version registry

Object	Structural payload version	Outer MPQFRAME version
HybridEnvelope	1	V2
EcPqEpochEnvelope	1	V2
SignedPQPublicKey	2	V2
DevicePublicBundle	2	V2
RoomKeyPayload	format-specific	V1
RoomInboundSession	1	V1
RoomMessageEnvelope	format-specific	V1

20.3 Downgrade rule

Production parsers MUST require the exact expected frame type, suite, and version. There is no general “try newest then fall back” negotiation in the canonical production path. Legacy helpers are migration/test surfaces and MUST NOT be exposed as transparent fallback in a production build.

20.4 Classical-layer downgrade

The SafeMeet core extends a vodozemac Olm context rather than defining the entire classical handshake. The final audit MUST verify that the host cannot reach an unintended older/weak classical path beneath the PQ layer and that the session identifier used in the hybrid transcript corresponds to the intended authenticated classical session.

21. Secret lifecycle and zeroization

21.1 Secret record contract

`E2eeSecretStore` defines typed secret records and canonical AAD binding user, device, record type, record ID/scope, and version. A production store MUST provide confidentiality and integrity, MUST NOT log plaintext secret records, and MUST authenticate the supplied AAD.

21.2 Zeroizing wrappers

Temporary message-key, shared-secret, and copied secret-byte buffers SHOULD use zeroizing containers. Debug formatting of secret wrapper types MUST be redacted.

21.3 Identity-key lifecycle

Long-term identity keys are intentionally retained for device lifetime and therefore are not covered by “consume once” semantics. Their creation, secure storage, migration, backup eligibility, revocation, and device-reset policy MUST be part of the host's device lifecycle design.

21.4 ML-KEM epoch key lifecycle

Current source removes a private epoch prekey from its map after successful committed use. The following remain unresolved at the protocol/deployment level: expiry, signed generation, number of simultaneously retained decapsulation keys, replenishment, crash-safe deletion, publication rollback, and exact server atomic-claim behavior. These are audit blockers for an unqualified ongoing-PQ-ratchet claim.

21.5 Deletion semantics

Logical deletion from a Rust container does not prove physical erasure from allocator pages, swap, crash dumps, SSD remapping, backup snapshots, or host storage. Audit claims SHOULD distinguish logical key retirement from platform-level secure deletion.

22. Error handling, resource bounds, and fail-closed behavior

22.1 Parser errors

Malformed magic, versions, suite/type fields, lengths, UTF-8, reserved bits, and trailing bytes MUST return errors rather than panic or reinterpret input.

22.2 Authentication errors

AEAD or signature failure MUST NOT advance replay or ratchet state. Error strings SHOULD avoid secret-dependent detail and SHOULD NOT leak key material.

22.3 Bounds

Attacker-controlled state is explicitly bounded for pending EC-PQ epoch messages, skipped classical keys, sparse-PQ skips, future epoch gap, room skips, replay segment entry count, replay segment size, and atomic-state batch sizes.

When a bound is reached, the system MUST fail in a confidentiality-preserving manner. The deployment SHOULD surface enough telemetry to diagnose availability attacks without recording plaintext or secret material.

22.4 Eviction policy

The specification distinguishes “reject new item” from “evict old authenticated item.” Silent eviction can cause permanent loss of legitimate delayed messages. Every bounded structure used in production SHOULD have a documented policy and corresponding test. Where the current implementation rejects rather than evicts, that behavior SHOULD remain explicit in the external specification.

23. Intended security properties and qualifications

23.1 Confidentiality and integrity

Assuming endpoint secrets remain confidential, identity verification is not bypassed, and XChaCha20-Poly1305/HKDF/SHA-256/underlying public-key primitives behave as expected, ciphertext modification or authenticated-context transplantation should fail before plaintext exposure.

23.2 Hybrid robustness

The intended per-message property is that the final AEAD key is derived from both EC and PQ message-key contributions under a domain-separated HKDF construction, so compromise of only one branch should not reveal the final key if the other input retains sufficient entropy.

AUDIT REQUIREMENT: this intended robustness and the surrounding state composition require independent cryptographic analysis; HKDF use alone does not prove the state-machine theorem.

23.3 Forward secrecy

Fresh X25519 material is generated per message on the EC branch, but old receiving private keys are retained within a configured bound to support reordering. Therefore compromise exposure depends on the precise retained-key state at compromise time; “perfect” or immediate erasure-based FS MUST NOT be claimed without quantifying this window.

23.4 Post-compromise security

Sparse ML-KEM epochs are intended to inject fresh post-quantum entropy after compromise. Whether recovery occurs against an active attacker depends on epoch transport, selective drop, retained decapsulation keys, and progress policy. Those issues remain open; therefore Level-3-style PCS language MUST be qualified until they are specified, modeled, and audited.

23.5 Authentication

Dual Ed25519 + ML-DSA signatures provide forward-looking hardening of device public material. They do not solve malicious first-contact directory substitution without an independent verification mechanism.

23.6 Group security

Group confidentiality and authenticity use a symmetric sender-key chain whose seed is distributed over eligible pairwise channels. A compromise of a current sender chain is recovered by room-session rotation, not by an ML-KEM operation per group message.

24. Wire-format registry

Frame type	Numeric ID	Expected outer version	Main source
HybridEnvelope	1	V2	src/protocol/hybrid_envelope.rs / src/session/hybrid_olm_mlkem.rs
EcPqEpochEnvelope	2	V2	src/protocol/ec_pq_epoch_envelope.rs / src/ratchet/ec_pq_epoch_ratchet.rs
SignedPQPublicKey	3	V2	src/identity/signed_pq_key.rs
DevicePublicBundle	4	V2	src/identity/device_bundle.rs
RoomKeyPayload	5	V1	src/group/room_session.rs
RoomInboundSession	6	V1	src/group/room_session.rs
RoomMessageEnvelope	7	V1	src/group/room_session.rs

Compatibility principle: the frame version identifies the outer protocol profile; structural payload versions independently protect each embedded layout. Both MUST be validated where present.

25. Domain-separation registry

The following registry is normative for the inspected profile. A domain string MUST NOT be reused for a semantically different purpose without a protocol version change. Initial-session domains use the SafeMeet-specific olm-mlkem v2 namespace. The EC-PQ Epoch Ratchet retains three historical general-sparse-triple domain strings and the historical triple-envelope replay prefix solely as compatibility identifiers for the already-established message-ratchet profile; these strings are not the current construction name and do not imply Signal provenance or transfer of proofs.

Purpose	Domain
Signed PQ public key	matrix-vodozemac-dual-signed-pq-public-key-mlkem768-v2
Device bundle	matrix-vodozemac-dual-signed-device-public-bundle-mlkem768-v2
Hybrid session fingerprint	safemeet-core-olm-mlkem-session-fingerprint-hash-v2
Hybrid session ID KDF	safemeet-core-olm-mlkem-session-id-v2
Initial payload key KDF	safemeet-core-olm-mlkem-initial-payload-key-v2
Initial payload AAD	safemeet-core-olm-mlkem-initial-payload-aad-v2
Hybrid AEAD helper AAD domain	matrix-vodozemac-pq-hybrid-aead-v1
EC message key	matrix-vodozemac-general-ec-ratchet-key-v1
PQ epoch chain key	matrix-vodozemac-general-sparse-pq-epoch-chain-key-v1
PQ message key	matrix-vodozemac-general-sparse-pq-message-key-v1
PQ next chain key	matrix-vodozemac-general-sparse-pq-next-chain-key-v1
EC-PQ hybrid extract salt	matrix-vodozemac-general-sparse-triple-hybrid-extract-salt-hkdf-sha256-v2

Purpose	Domain
EC-PQ hybrid expand info	matrix-vodozemac-general-sparse-triple-hybrid-expand-info-hkdf-sha256-v2
EC-PQ epoch message AAD	matrix-vodozemac-general-sparse-triple-aad-mlkem768-v2
Initial replay hash	safemeet-core-replay-initial-olm-mlkem-envelope-hash-v2
EC-PQ epoch replay session hash	matrix-vodozemac-replay-triple-envelope-session-id-hash-v1
Room-key-event replay hash	matrix-vodozemac-replay-room-key-event-hash-v1
Replay segment checksum	matrix-vodozemac-replay-segment-sha256-v1
Room session ID	matrix-vodozemac-room-session-id-mlkem768-v1
Room inbound-session storage	matrix-vodozemac-room-inbound-session-store-mlkem768
Room message key	matrix-vodozemac-room-message-key-mlkem768-v1
Room next chain	matrix-vodozemac-room-next-chain-key-mlkem768-v1
Room message AAD	matrix-vodozemac-room-message-aad-mlkem768-v1
Trust-store ML-DSA identity hash	matrix-vodozemac-trust-store-mldsa-identity-public-key-hash-v1
Identity-verification ML-DSA hash	safe-e2ee-core-phase8-mldsa-identity-public-key-hash-v1
Safety number	safe-e2ee-core-safety-number-v1
Verified room-state event	safe-e2ee-verified-room-state-event-v1
Verified room-state hash	safe-e2ee-verified-room-state-hash-v1
Backup KDF	safe-e2ee-core-backup-envelope-kdf-v1
Backup AAD	safe-e2ee-core-backup-envelope-aad-v1
E2EE atomic state batch checksum	matrix-vodozemac-e2ee-atomic-state-batch-sha256-v1
Production receive state-batch marker	matrix-vodozemac-production-receive-state-batch-sha256-v1
Anti-rollback marker	safe-e2ee-core:anti-rollback-marker:v1

A machine-generated registry SHOULD be produced from source before the final audit to ensure that no additional domain labels were omitted and no duplicates exist unintentionally.

26. Test vectors and implementation assurance

26.1 Current test inventory

A static scan of the v1.0 source finds 552 #[test] attributes across src/ and tests/. Nine of these reside in the non-compiled src/legacy/ tree. The active source inventory therefore contains 543 test functions before Cargo target/feature filtering (200 under active src/ and 343 under tests/). Authoritative executed/pass/ignore counts are taken from the frozen-release evidence logs, not from this static inventory.

26.2 Current fuzz targets

The source contains four parser/envelope cargo-fuzz targets plus one dedicated stateful EC-PQ ratchet fuzz target:

```
protocol_frame_fuzz.rs
backup_envelope_fuzz.rs
room_key_envelope_fuzz.rs
verified_room_state_fuzz.rs
```

The four targets under fuzz/ provide parser/envelope coverage. The stateful-fuzz target drives long EC-PQ operation traces, and integration tests add randomized replay-aware traces and persistence fault injection. Sustained campaign duration, coverage, corpus, and crash-triage evidence must still be generated for a frozen release.

26.3 Current deterministic vectors

The current vector directory includes vectors for signed PQ keys, device bundles, hybrid envelopes/AAD, EC-PQ epoch envelope versions, EC-PQ hybrid KDF/AAD, replay keys, room KDF/AAD, room inbound session storage, and room-key payload compatibility.

At minimum, the audit package SHOULD publish stable vectors for:

- each canonical frame type and negative mutation class;
- both signatures over SignedPQPublicKey and DevicePublicBundle transcripts;
- hybrid setup transcript/session-ID/initial-key/AAD;
- EC-PQ hybrid KDF and AAD with and without PQ update;
- out-of-order and replay decisions;
- room message KDF/AAD;
- backup KDF/AAD and tamper negatives;
- atomic state-batch and replay-segment parsing.

26.4 Reference implementation

The Rust crate is the current reference implementation. For auditability, a smaller non-production reference implementation or executable vector generator SHOULD be created for the cryptographic transcript/KDF functions. It SHOULD avoid duplicating the same helper functions from production code so that independent cross-checking is meaningful.

27. Source-to-protocol mapping

Protocol function	Primary source modules
Canonical encoding/frame	src/protocol/protocol_encoding.rs, src/protocol/protocol_frame.rs
ML-KEM wrapper	src/crypto/pq.rs
ML-DSA identity	src/crypto/mldsa_identity.rs
Nested signed PQ key	src/identity/signed_pq_key.rs
Device bundle	src/identity/device_bundle.rs
Olm-ML-KEM Session Binding	src/session/hybrid_olm_mlkem.rs, src/crypto/hybrid_aead.rs, src/crypto/kdf.rs
EC ratchet	src/ratchet/ec_message_ratchet.rs
Sparse PQ ratchet	src/ratchet/general_sparse_pq.rs
EC-PQ Epoch Ratchet	src/ratchet/ec_pq_epoch_ratchet.rs, src/protocol/ec_pq_epoch_envelope.rs
Replay	src/state/replay_protection.rs
Persistence transaction	src/state/e2ee_state_transaction.rs
Secret storage	src/state/e2ee_secret_store.rs
Anti-rollback	src/state/anti_rollback.rs, src/state/e2ee_rollback_store.rs
Trust/evidence	src/trust/trust_store.rs, src/identity/identity_verification.rs

Protocol function	Primary source modules
Room/session cryptography	src/group/room_session.rs
Room-key distribution	src/group/room_key_distribution.rs, src/group/room_group_policy.rs
Verified room state	src/trust/verified_room_state.rs
Backup/export	src/backup/backup_envelope.rs
Production policy	src/runtime/production.rs, src/runtime/e2ee_policy.rs, src/lib.rs

28. Audit gap register

The following register merges the whitepaper, the informal third-party technical review, and current source inspection. Priority refers to audit readiness, not necessarily exploitability.

ID	Priority	Area	Current status	Required closure/evidence
AR-01	P0	PQ epoch transport under selective drop	OPEN	Define epoch trigger, retransmission/progress, outstanding epochs, adversarial-drop behavior; model it.
AR-02	P0	Final hybrid combiner	IMPLEMENTED; audit still required	HKDF-SHA256 V2 + KAT exists. Obtain independent composition review/formal argument.
AR-03	P0	Formal ratchet model	PARTIAL	Initial Tamarin models and a model-to-Rust map exist. Execute/refine the models, preserve proof logs/counterexamples, and obtain independent correspondence review before making formal-verification claims.
AR-04	P0	ML-KEM decapsulation-key lifecycle	OPEN	Specify retention bound, expiry, consumption, retirement, replenishment, crash recovery, zeroization.
AR-05	P0	vodozemac/classical boundary	EVIDENCE REQUIRED	Pin exact source; verify all-zero behavior, reachable protocol versions, session-ID semantics, local patches.
AR-06	P1	Initial handshake replay mandatory	RESOLVED for production wrapper	Ensure product uses only replay-aware high-level API; prevent unsafe low-level use in production integration.
AR-07	P0/P1	Atomic receive persistence	PARTIAL	Integrate actual ratchet+replay bytes into one durable transaction or prove host equivalent; fault-inject.

ID	Priority	Area	Current status	Required closure/evidence
AR-08	P1	First-contact authenticity	HOST REQUIREMENT	Production KT/cross-signing/safety-number policy; high-assurance mode must not rely on TOFU alone.
AR-09	P1	Stateful ratchet fuzzing	PARTIAL	A dedicated stateful fuzz target plus randomized trace and persistence fault-injection tests exist. Run sustained campaigns with corpus, coverage, duration, crash triage, and CI evidence.
AR-10	P1	Security invariants	PARTIAL	Written invariants exist; connect to model/fuzzer and CI assertions.
AR-11	P2	ML-KEM/ML-DSA category mismatch and bundle size	DOCUMENTATION OPEN	Record rationale and measured production bundle/key/signature sizes.
AR-12	P2	Bounded-state availability behavior	PARTIAL	Document reject/eviction behavior and per-session/per-peer limits; add availability tests.
AR-13	P2	Legacy parser exposure	MOSTLY RESOLVED	Verify production feature matrix in CI; consider explicit legacy-only feature for all helpers.
AR-14	Accepted	Group claim discipline	CORRECT	Keep sender-key + pairwise PQ wording; define rotation/recovery policy.
AR-15	P2	Protocol naming / provenance	RESOLVED	Current code, API, vectors, whitepaper, and specification use SafeMeet Olm-ML-KEM Session Binding and SafeMeet EC-PQ Epoch Ratchet. Historical compatibility identifiers do not imply transferred provenance or proofs.
AR-16	P0	Outbound ratchet crash atomicity	OPEN	Staged send state + durable commit + idempotent publish/retry; fault-injection tests.
AR-17	P1	Local journal authenticity	HOST REQUIREMENT	Do not rely on unkeyed checksums against malicious writer; use authenticated secret store + monotonic source.
AR-18	P1	PQ receive-epoch retention bound	OPEN/PARTIAL	Specify/implement bounded retirement of receive epoch state and prove delayed-message behavior.

ID	Priority	Area	Current status	Required closure/evidence
AR-19	P1/P2	Room transport PQ assertion	PARTIAL	Replace/strengthen `is_pq_protected()` Boolean with verifiable capability/concrete integration evidence.
AR-20	P2	Windows/non-Unix crash durability	OPEN EVIDENCE	Target-platform crash/power-loss tests; document actual rename/directory durability semantics.

29. Formal-model requirements and invariants

29.1 Model scope

The highest-value formal model is the one-to-one state machine, including initial hybrid setup, EC ratchet, sparse PQ epochs, buffering, replay, authentication failure, and compromise events. It need not model file I/O byte-for-byte initially; persistence can be represented as atomic/failed commit actions and refined later.

29.2 Minimum state variables

The model SHOULD represent:

- local/peer identity and expected subject;
- classical session identifier;
- EC send/receive counters and retained keys;
- PQ epoch identifiers, message indexes, prekey IDs, and retained decapsulation keys;
- pending messages and skipped keys;
- replay set;
- persistent generation/version;
- compromise events for classical state, PQ state, and full endpoint state;
- network actions: deliver, duplicate, reorder, delay, drop, mutate.

29.3 Required invariants

At minimum, the model and stateful fuzzer SHOULD assert:

```
I1  Authentication failure never advances long-lived ratchet state.
I2  Authentication failure never adds an object to committed replay state.
I3  A buffered/PendingEpoch object is not committed as replayed.
I4  One replay identity causes plaintext exposure at most once.
I5  Committed message/epoch counters are monotonic except through explicitly detected rollback.
I6  Pending/skipped/retained attacker-influenced state remains within configured bounds.
I7  A private PQ prekey is not consumed before the authenticated epoch transition commits.
I8  Wrong expected user/device identity cannot become an accepted bundle solely through self-
identification.
I9  Blocked/revoked/removed devices cannot receive current room keys through ordinary refresh.
I10 Production canonical parsing never falls back to legacy on malformed canonical input.
I11 Plaintext is not returned when required persistence commit fails.
I12 No send retry reuses an AEAD nonce/key/state combination after a crash or partial commit.
```

29.4 Security queries

The audit should ask separately:

- Does secrecy hold if the EC branch is compromised but the PQ branch remains secure?
- Does secrecy hold if the PQ branch is compromised but the EC branch remains secure?
- What is the exact compromise window created by retained EC and PQ keys?
- After a full state compromise, under what delivery assumptions does a fresh PQ epoch restore secrecy?
- Can selective dropping indefinitely suppress PQ recovery while classical traffic continues?
- Can delayed epoch material cause future keys to be retained longer than intended?

30. Stateful fuzzing and fault-injection requirements

30.1 Ratchet trace fuzzer

A new state-machine fuzz target SHOULD generate long operation sequences, not isolated byte strings. Operations should include send, receive, duplicate, delay, reorder, mutate, inject future epoch, omit epoch update, retry, crash before commit, crash after state-file write but before rename, replay after restart, device compromise marker, and session reload.

After every operation, the harness SHOULD assert the invariants in Section 29 and compare sender/receiver logical states where applicable.

30.2 Differential testing

Where practical, transcript and KDF outputs SHOULD be cross-checked against a small independent implementation. Parser output SHOULD round-trip canonical bytes and reject all non-canonical mutations.

30.3 Persistence fault injection

Fault injection SHOULD cover at least:

- temporary file created but empty;
- short write;
- file synced but rename absent;
- rename completed but directory metadata not durable;
- replay segment committed but state batch absent;
- state batch committed but replay segment absent;
- process restart between every commit stage;
- concurrent writers to same session/replay scope;
- monotonic CAS failure;
- restored older file snapshot with newer monotonic counter.

30.4 Fuzz CI evidence

Audit readiness requires not only fuzz target source but evidence of campaign duration, corpus, sanitizer/coverage settings, crash triage, and a reproducible CI or scheduled job.

31. Audit engagement scope

A final specialist cryptographic engagement SHOULD be scoped around four primary questions rather than a generic code review.

31.1 Hybrid composition

Does the final construction achieve the intended property that breaking only the EC branch or only the ML-KEM branch does not reveal the message key, considering exact context binding and state transitions?

31.2 Forward secrecy and PCS across the epoch cycle

Do the stated forward-secrecy and post-compromise-security properties hold before, during, and after epoch changes under both passive and active adversarial scheduling, including in-flight retained decapsulation keys?

31.3 Long-trace state-machine soundness

Can an adversary reach an invalid state through reordering, duplicates, pending queues, authentication failures, retries, crashes, or concurrent commits? Are at-most-once commit, monotonicity, and bounded-state properties preserved?

31.4 Classical trust boundary and downgrade

Does the vodozemac/Olm boundary preserve the assumptions the PQ extension makes, including session identifier binding, unsupported legacy path rejection, and X25519 edge cases?

A separate implementation workstream SHOULD review constant-time behavior, secret zeroization, allocator lifetime, and platform storage semantics.

32. Evidence package and release gates

32.1 Minimum auditor package

Before final audit engagement, provide:

- this specification and whitepaper;
- frozen source archive with SHA-256 and exact dependency lockfiles;
- source manifest and build-feature matrix;
- current audit gap register with owner/status;
- all canonical/negative vectors and a vector generator;
- full test/CI output for the frozen archive;
- fuzz target sources, corpora, run logs, and crash history;
- symbolic model and correspondence map to Rust modules;
- dependency/advisory report;
- clippy/rustfmt/build logs;
- target-platform persistence fault-injection evidence;
- production `E2eeSecretStore` and monotonic-store integration evidence if deployment claims secure persistence;
- Key Transparency/cross-signing integration evidence if deployment claims first-contact server-substitution resistance.

32.2 Release gates

A broad production release SHOULD be blocked unless:

1. P0 audit gaps have an implemented and tested closure or a documented security-claim downgrade accepted by the security owner.
2. Full production-feature test suite passes on the exact frozen archive.
3. Clippy with warnings-as-errors and formatting checks pass.
4. Fuzz smoke and scheduled stateful fuzz campaigns show no unresolved crashes.
5. Dependency/advisory review has no unaccepted high-severity findings.
6. Production secret storage is authenticated and reviewed.
7. If rollback resistance is claimed, the monotonic store is deployed and tested.
8. The host uses only production-safe identity/replay/trust APIs.
9. Room-key transport and rotation policy are concretely verified.
10. Audit exceptions and residual risks are signed off and published with the release evidence.

33. Known limitations and unresolved questions

The following limitations remain explicit at this document version:

- The custom hybrid ratchet has not yet been formally verified or independently audited.
- Active-adversary PQ recovery depends on an underspecified epoch transport/progress policy.
- ML-KEM decapsulation-key retention, expiry, generation, and replenishment are not completely specified in the current source.
- Outbound ratchet persistence is not yet a fully staged atomic send transaction.
- Receive-side atomic persistence primitives exist, but the strongest one-batch ratchet+replay guarantee is not fully integrated into the production wrapper.
- Local replay/state journal checksums are not keyed authentication against a malicious disk writer.
- True rollback resistance depends on a host monotonic store.
- Key Transparency and cross-signing backends are interfaces rather than complete production services.
- TOFU cannot detect malicious first contact.
- The group protocol is sender-key based and lacks per-message PQ group PCS; recovery depends on room-session rotation.
- The concrete group rotation policy is not specified in the core.
- The PQ-protected room transport check is currently an implementation-reported Boolean rather than an unforgeable protocol capability.
- Windows/non-Unix directory metadata durability requires empirical target-platform validation.

- An initial stateful EC-PQ ratchet fuzz harness is present at stateful-fuzz/fuzz_targets/ec_pq_epoch_trace_fuzz.rs, together with randomized long-trace and persistence fault-injection integration tests. Sustained fuzz-campaign evidence, coverage measurements, corpus history, and long-duration crash-free execution logs remain incomplete.
- Metadata privacy, endpoint compromise, traffic analysis, and product recovery UX are outside the core protocol.

These limitations are part of the protocol's security posture and MUST remain visible until evidence supports their removal.

Appendix A. State-transition tables

A.1 Initial setup receive

State/action	Condition	Next action	Commit allowed?
Parse frame	canonical V2 HybridEnvelope	verify expected identities	No
Verify bundles	all nested/outer signatures and expected subjects valid	decapsulate ML-KEM	No
Derive transcript/key	exact classical session and bundle context	AEAD open	No
AEAD failure	any tamper/context mismatch	error	No
AEAD success	plaintext authenticated	stage replay	Not yet
Replay commit failure	storage/concurrency failure	zeroize plaintext, error	No
Replay commit success	first accepted object	return plaintext	Yes

A.2 EC-PQ epoch-ratchet receive

State/action	Condition	Result
Parse	bad version/suite/type/length	Error; no state change
Replay precheck	already committed	Replay error
Future bounds	epoch/message too far ahead	Error; bounded state
PQ prepare	required epoch missing	Pending; no replay commit
PQ prepare	update available	tentative decapsulation/state
EC prepare	candidate receiving key found	tentative EC state
AEAD	failure	discard tentative EC/PQ state
AEAD	success	stage persistence/replay
Persistence	failure	no plaintext exposure
Commit	success	commit in-memory state, expose plaintext

A.3 Room receive

Check	Required behavior
Wrong room/session/sender/device	Reject
Duplicate seen message index	Reject replay
Gap within bound	Derive/store skipped keys and authenticate
Gap beyond bound	Reject without unbounded work
AEAD failure	No chain-state commit
AEAD success	Commit next chain/seen state, then expose plaintext according to persistence contract

Appendix B. Reference test commands

The audit package SHOULD capture exact output for the frozen source. Example one-line PowerShell commands for the project's configured PQ backend are:

```
$env:RUST_BACKTRACE="1"; cargo test --features mlkem-pqcrypto --all-targets -- --nocapture
cargo test --features mlkem-pqcrypto --doc -- --nocapture
```

The repository's audit documentation also defines a production profile using `--no-default-features --features production`; CI SHOULD select one canonical release command and record the feature expansion so reviewers know exactly which code was compiled.

Authoritative Windows build/test evidence for the frozen v1.0 core snapshot is included in the release evidence package. The exact sibling vodozemac dependency is also supplied as vodozemac.zip. Because the core archive itself contains no .git metadata, Git branch/clean-tree claims are not made for that archive; archive SHA-256 and the full file manifest provide the release binding.

Appendix C. Change-control rules

The following changes require a protocol-version or explicit compatibility review and new test vectors:

- any change to byte order, LP encoding, frame header, structural payload format, or domain string;
- any change to the order or representation of signed transcript fields;
- any KDF salt/IKM/info/context change;
- any AEAD AAD field change;
- any change to which state transition occurs before/after authentication or persistence;
- any change to replay identity construction;
- any change to ML-KEM prekey consumption/retention semantics;
- any change to room-key recipient authorization or room-state epoch semantics;
- any change that permits a production parser to accept a previously rejected legacy/unknown form.

For every protocol-sensitive change, the pull request SHOULD include: specification diff, test-vector diff, negative tests, state-machine invariant impact, migration plan if wire compatibility changes, and explicit security-review signoff.

Normative conclusion

SafeMeet's E2EE core contains a substantial implementation of hybrid classical/post-quantum identity, SafeMeet Olm-ML-KEM Session Binding, the SafeMeet EC-PQ Epoch Ratchet, canonical framing, replay control, room sender keys, and encrypted backup primitives. The current source uses a domain-separated HKDF-SHA256 v2 final EC-PQ message-key combiner, replay-aware production APIs, and atomic-state building blocks. SafeMeet-specific protocol names are used without claiming that Signal PQXDH, Triple Ratchet, or SPQR proofs transfer to SafeMeet.

Audit readiness, however, requires more than the presence of cryptographic primitives. Before making unqualified ongoing-PQ PCS or crash-safe persistence claims, the project must close or explicitly downgrade the claims around PQ epoch transport under adversarial drop, ML-KEM decapsulation-key lifecycle, outbound send atomicity, fully integrated receive persistence, the vodozemac boundary, sustained stateful-ratchet fuzzing evidence, and executed/refined formal state-machine analysis. First-contact authenticity and true local rollback resistance remain deployment-level responsibilities unless their real backends are included in the audited system.

This specification is therefore both a protocol description and a boundary against overclaiming: implemented behavior is stated precisely, required host assumptions are explicit, and unresolved properties remain visible as audit work rather than being implied by architecture alone.

References and evidence basis

1. SafeMeet project, safe-e2ee-core-production-v1.0.zip, implementation snapshot identified by SHA-256 in Document Control.
2. SafeMeet Core E2EE Security Whitepaper, Version 1.0 (Initial Project Version), 15 August 2026.
3. SafeMeet Core E2EE, informal third-party technical review of the earlier whitepaper; not an audit.
4. NIST FIPS 203, *Module-Lattice-Based Key-Encapsulation Mechanism Standard (ML-KEM)*, 2024.
5. NIST FIPS 204, *Module-Lattice-Based Digital Signature Standard (ML-DSA)*, 2024.
6. Krawczyk and Eronen, RFC 5869, *HMAC-based Extract-and-Expand Key Derivation Function (HKDF)*.
7. Signal, *The ML-KEM Braid* protocol specification and Sparse Post-Quantum Ratchet reference material, as cited by the pre-audit review.

8. Eurocrypt 2025 and USENIX Security 2025 analyses of post-quantum ratcheting, as cited by the pre-audit review.
9. Apple Security Engineering, *iMessage with PQ3*, and accompanying independent security analysis, as cited by the pre-audit review.

Where an external paper or standard is used in the final audit, the auditor SHOULD cite the exact version/DOI/URL in the engagement report rather than relying on this abbreviated evidence list.