SafeMeet Core E2EE Security

Technical Whitepaper for the Post-Quantum Cryptographic Core

Core E2EE Edition

Version: 1.0 (Initial Project Version)
Date: August 15, 2026
Implementation basis: safe-e2ee-core-production-v1.0.zip
Reference source SHA-256: c88928d64b07cd0789481b4f21ba9a7fe405df20dc8589158853bd75dc383210

Scope boundary
This paper describes the cryptographic E2EE core: identity and device bundles, hybrid session establishment, one-to-one ratcheting, envelope framing, replay protection, trust verification, group cryptography, backup-envelope cryptography, fail-closed policy, and implementation testing. General server, transport, UI, and product workflows are intentionally excluded except where they define cryptographic context.

Document scope: technical whitepaper describing the SafeMeet E2EE algorithms, protocol architecture, implementation structure, and corresponding source-code modules.

Executive Summary

SafeMeet implements a substantial post-quantum end-to-end encryption core built around hybrid classical and post-quantum cryptography. Device identity is authenticated using Ed25519 and ML-DSA-44. ML-KEM-768 contributes post-quantum shared secrets during initial session setup and during later sparse post-quantum ratchet epochs. One-to-one messages use a SafeMeet EC-PQ Epoch Ratchet that combines a classical X25519 message-ratchet branch with a sparse ML-KEM epoch branch, and encrypts payloads with XChaCha20-Poly1305.

A central implementation property is the receive-side commit discipline. EC and PQ receive steps are prepared tentatively, the ciphertext is authenticated, and only then is the cryptographic state committed. Together with bounded skipped keys, bounded pending queues, duplicate-pending rejection, and a maximum future-epoch gap, this reduces the risk of state poisoning and resource-amplification attacks.

The identity layer uses two nested authentication boundaries. SignedPQPublicKey binds an ML-KEM public key to device and algorithm context with both classical and post-quantum signatures. DevicePublicBundle then signs the complete device bundle again. Verification requires caller-supplied expected user and device identities, reducing the risk of accepting a valid bundle for the wrong subject.

Canonical MPQFRAME framing binds protocol version, cipher suite, and frame type, rejects non-zero reserved fields, validates lengths, and rejects trailing bytes. Production feature gates disable legacy parsing paths and reject incompatible feature combinations at compile time. Therefore, legacy compatibility is principally a non-production and migration concern rather than an exposed production parser path in the inspected build configuration.

Group messaging uses a sender-key room session. Room keys are distributed through PQ-protected one-to-one E2EE channels, and verified room-state logic binds active membership, room epoch, member epoch, event ordering, and trusted signers. This is not a full post-quantum group key agreement protocol, and group post-compromise recovery remains dependent on room-key rotation rather than per-message group ratcheting.

This whitepaper presents the first technical description of the SafeMeet E2EE core and its reference implementation. The focus is the design and composition of the cryptographic algorithms, the protocol state transitions, canonical encodings, replay and ordering behavior, trust controls, group sender-key construction, backup-envelope cryptography, and the Rust source modules that implement them. The document also states the current technical limitations and directions for future development.

Contents

1. Scope and Document Overview

This section defines the scope of the first SafeMeet E2EE core version and the relationship between the protocol description and the accompanying source implementation.

The whitepaper describes the cryptographic behavior implemented by the SafeMeet E2EE core, including public-key identity, signed device bundles, hybrid session setup, ratchet state transitions, message envelopes, AEAD context binding, replay logic, trust decisions, group sender-key cryptography, backup-envelope cryptography, fail-closed policy, and implementation testing.

The document intentionally does not design general messaging infrastructure. References to an “adversarial delivery path” are threat-model assumptions used to explain parser, replay, ordering, and state-transition behavior; transport, server, account, and user-interface systems remain outside the E2EE core described here.

> Initial project version
> This whitepaper documents version 1.0 of the SafeMeet E2EE core and the corresponding Rust reference implementation. The document focuses on the algorithms, protocol structure, source-code organization, and technical limitations of this initial release.

1.1 In-Scope Components

Core area	Primary responsibility
Identity and bundles	Bind classical and post-quantum public keys to an expected user and device identity.
Hybrid setup	Combine classical Olm session context with ML-KEM shared secret material and encrypt the initial payload.
One-to-one ratchet	Derive per-message keys from EC and sparse PQ branches and evolve receive state safely.
Replay and ordering	Reject repeated envelopes, bound out-of-order state, and avoid invalid-ciphertext state poisoning.
Framing and parsing	Provide canonical, versioned, cipher-suite-bound binary envelopes with strict decoding.
Trust and verification	Pin identities, detect replacement, derive safety numbers, and evaluate stronger evidence.
Group cryptography	Encrypt room messages with sender keys and distribute room keys over PQ-protected one-to-one E2EE.
Backup envelope	Encrypt and authenticate typed E2EE records under a high-entropy recovery key.
Policy and implementation	Enforce fail-closed production settings and provide tests, fuzz targets, zeroization, redaction, and strict integration entry points.

1.2 Document Status

This paper is implementation-grounded. Algorithms and data flows are described together with the Rust modules that implement them. Where functionality depends on a host application or remains incomplete in the initial project version, that limitation is stated explicitly. The whitepaper is a technical description of the design and source implementation, not a formal proof of the protocol.

2. Security Objectives and Terminology

2.1 Core Security Objectives

- Confidentiality: message plaintext and secret key material should be recoverable only by intended endpoints.
- Integrity and context binding: modification of ciphertext or authenticated metadata must cause decryption failure.

- Hybrid security: the message key should depend on both classical and post-quantum inputs so that compromise of only one branch is insufficient under the intended composition.
- Post-quantum protection: ML-KEM-768 should contribute secrecy at session establishment and later sparse PQ epochs; ML-DSA-44 should contribute long-term authenticity of device key material.
- Identity substitution resistance: public keys must be bound to caller-expected user, device, and algorithm context.
- Replay and reordering resistance: duplicate, delayed, reordered, and future-epoch messages must be handled without silent acceptance or unbounded state growth.
- State-poisoning resistance: invalid ciphertext must not advance long-lived ratchet state.
- Downgrade resistance: production code must fail closed on non-canonical frames, unsupported cipher suites, identity changes, and missing PQ material.

2.2 Terminology

Term	Meaning in this paper
E2EE	End-to-end encryption performed by endpoint cryptographic state; ordinary message plaintext is not part of the wire envelope.
ML-KEM-768	Post-quantum key encapsulation mechanism used to derive shared secret material.
ML-DSA-44	Post-quantum digital signature scheme used to authenticate identity and device-bundle material.
SafeMeet Olm-ML-KEM Session Binding	SafeMeet's hybrid setup that combines verified ML-KEM material with an existing classical Olm session context.
SafeMeet EC-PQ Epoch Ratchet	One-to-one construction combining an EC message-ratchet branch and a sparse PQ epoch branch.
Epoch	A PQ interval in which an ML-KEM update establishes an epoch chain key and per-message PQ keys are derived.
AAD	Additional Authenticated Data: metadata authenticated by AEAD but not encrypted.
TOFU	Trust On First Use: the first accepted identity is pinned, and later changes fail closed.
Safety number	Human-comparable fingerprint derived from local and peer identity material.
Room session	Sender-key chain used by one sender to encrypt a sequence of group messages.
Commit-after-AEAD	Prepare receive state, authenticate ciphertext, and commit the prepared state only after successful AEAD verification.

A comprehensive glossary of the terminology and concepts used throughout this whitepaper is provided in Appendix B.

3. Core Architecture

The core is divided into narrow layers so that identity, key agreement, ratcheting, parsing, replay, and trust can fail independently and explicitly.

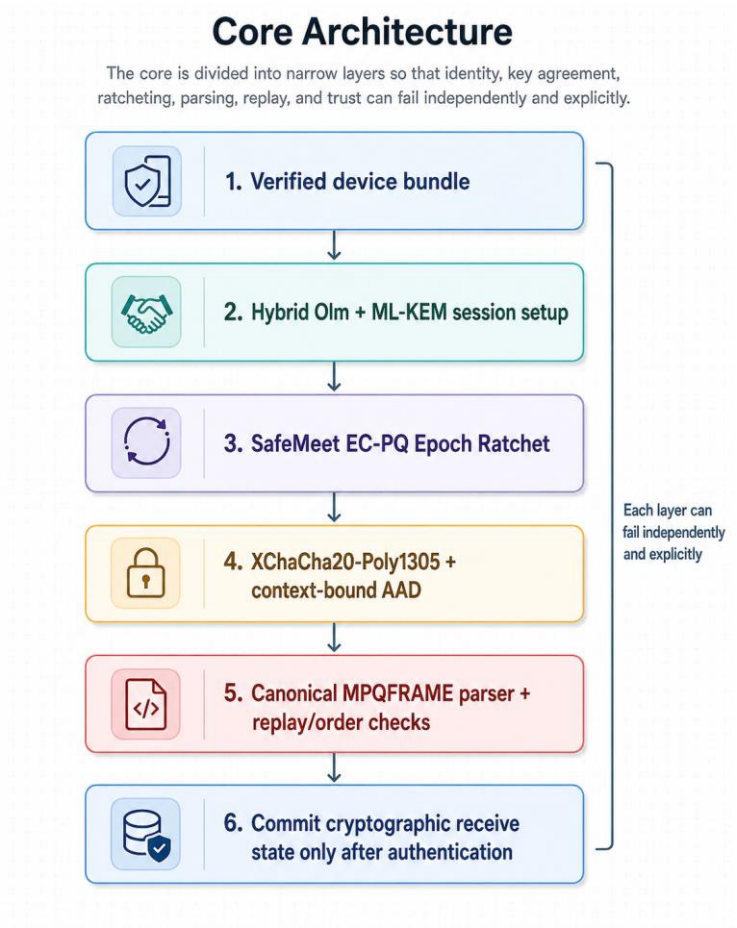

Figure 1. Core Architecture

3.1 Architectural Principles

- Authenticate before using key material. Device bundles and nested PQ keys are verified against externally expected identities before cryptographic use.
- Bind context into every security boundary. User, device, session, epoch, message, room, algorithm, and frame metadata are included in signed payloads, key derivation, replay identifiers, or AEAD AAD.
- Separate parse from policy. Canonical parsers validate binary structure; production policy separately rejects legacy input and unsupported suites.
- Stage before commit. Receive branches calculate tentative next state, authenticate the ciphertext, and commit only after success.
- Bound attacker-controlled state. Skipped keys, pending messages, and future epoch gaps are limited.
- Fail closed on trust regressions. Identity changes, stale or mismatched evidence, and unsupported room-key flows are rejected.

3.2 Core Layer Map

Layer	Representative modules	Role
Identity	device_bundle.rs; signed_pq_key.rs; mldsa_identity.rs	Dual-signature identity and PQ-key binding.
Session setup	hybrid_olm_mlkem.rs; hybrid_envelope.rs; pq.rs	Hybrid ML-KEM/classical setup and initial encrypted payload.
One-to-one ratchet	ec_message_ratchet.rs; general_sparse_pq.rs; ec_pq_epoch_ratchet.rs	Per-message EC/PQ key evolution and staged decrypt.
Replay	replay_protection.rs	Domain-separated initial and message replay identifiers.
Framing	protocol_frame.rs; protocol_encoding.rs; hybrid_aead.rs	Canonical frame, strict encoding, and AEAD helpers.
Trust	trust_store.rs; trust_manager.rs; identity_verification.rs	TOFU, safety numbers, evidence evaluation, and fail-closed trust state.

Layer	Representative modules	Role
Group	room_session.rs; room_key_distribution.rs; verified_room_state.rs; room_group_policy.rs	Sender-key rooms, verified room state, and key-sharing policy.
Backup	backup_envelope.rs	AEAD export/import envelope for typed E2EE records.
Policy	e2ee_policy.rs; production.rs; lib.rs	Production profile, strict wrappers, and compile-time guards.

4. Cryptographic Threat Model

The core assumes that all wire-facing inputs and downloaded public key material may be adversarial. An attacker may replay, reorder, duplicate, truncate, mutate, substitute, or cross-context transplant envelopes and public bundles. The parser and cryptographic state machine must reject such inputs without revealing plaintext or advancing state incorrectly.

4.1 Adversary Capabilities

- Supply malformed, truncated, oversized, non-canonical, or trailing-byte protocol inputs.
- Replay an initial Olm-ML-KEM binding envelope or a previously accepted one-to-one or room message.
- Reorder valid messages, delay an epoch update, or send a bounded-future message before its required PQ epoch is available.
- Replace individual keys inside a bundle, replace the complete bundle, or change algorithm identifiers.
- Present a valid bundle for the wrong user or device identity.
- Tamper with PQ ciphertext, EC public keys, message numbers, epochs, indexes, or AAD-bound room metadata.
- Attempt downgrade through legacy payloads, alternate cipher suites, missing PQ material, or non-PQ room-key distribution.
- Attempt to exhaust skipped-key or pending-message state through large gaps or duplicate inputs.

4.2 Out-of-Scope Compromise

A fully compromised endpoint can read plaintext and live key material and is outside the protection boundary of the cryptographic protocol. The core also does not claim metadata hiding, traffic-analysis resistance, or protection against a malicious authorized recipient. These exclusions do not reduce the requirement that the core reject hostile protocol inputs and preserve state-machine integrity.

5. Primitives, Key Derivation, and Domain Separation

5.1 Primitive Suite

Purpose	Primitive
Post-quantum key establishment	ML-KEM-768
Post-quantum signatures	ML-DSA-44
Classical signatures / identity	Ed25519
Classical DH ratchet material	X25519
Payload encryption	XChaCha20-Poly1305
Key derivation and transcript hashing	HKDF-SHA-256 in major setup/envelope paths and in the domain-separated v2 final EC-PQ epoch ratchet hybrid combiner
Secret handling	Zeroizing / explicit zeroize for intermediate key material

5.2 Domain Separation

Distinct domain strings separate device-bundle signatures, signed PQ keys, Olm-ML-KEM binding derivation, initial payload encryption, one-to-one message derivation, room message derivation, replay identifiers, safety numbers, and backup envelopes. Length-prefixing is used in several composite encodings to reduce ambiguity between variable-length fields.

5.3 Hybrid Composition Qualification

The SafeMeet Olm-ML-KEM Session Binding and backup paths use HKDF-SHA-256. The SafeMeet EC-PQ Epoch Ratchet uses an explicit domain-separated HKDF-SHA-256 v2 final combiner: the EC and PQ message-key contributions are length-prefixed into IKM, public ratchet context is bound into both the extract salt and expand info, and XChaCha20-Poly1305 authenticates the corresponding envelope metadata. This construction is part of the initial SafeMeet protocol profile and is covered by deterministic known-answer tests in the source repository.

> Design note
> The individual primitives are standardized or widely used; the SafeMeet-specific contribution lies in their composition, transcript and context binding, ratchet state transitions, replay handling, and integration into a single E2EE core.

6. Device Identity and Public Bundles

SafeMeet binds device identity to both classical and post-quantum public material. The design uses nested authentication rather than relying on a single signature over an informal serialization.

6.1 SignedPQPublicKey

- Contains the ML-KEM public key and algorithm context.
- Binds the key to the intended device context.
- Carries both Ed25519 and ML-DSA signatures.
- Includes algorithm identifiers in the signed data, preventing silent algorithm substitution.

6.2 DevicePublicBundle

- Contains classical identity and prekey material, Ed25519 identity, ML-DSA identity, signed ML-KEM material, and related public context.
- Signs the complete bundle at a second layer with Ed25519 and ML-DSA.
- Checks consistency between nested and outer identity material.
- Verifies against an ExpectedDeviceIdentity supplied by the caller rather than trusting only the bundle's embedded user_id and device_id.

6.3 Security Properties

Attack	Mitigation
Replace only the ML-KEM key	Nested dual signatures fail because the signed PQ key changes.
Replace another field in the bundle	Bundle-level dual signatures fail.
Swap the advertised algorithm	Algorithm identifiers are part of signed context.
Serve a valid bundle for the wrong subject	verify_expected compares the bundle with caller-supplied user and device identifiers.
Replace the complete identity after pinning	TOFU trust state detects the changed Ed25519 key or ML-DSA identity hash and fails closed.

The remaining structural limitation is first-contact authenticity. A completely substituted but internally valid bundle can still be accepted on first use unless the user or application supplies stronger evidence such as a verified safety number, cross-signing result, or key-transparency result.

7. SafeMeet Olm-ML-KEM Session Binding

SafeMeet Olm-ML-KEM Session Binding is a SafeMeet-specific construction that binds verified ML-KEM-768 material and both endpoint device bundles to an already-established classical Olm session context. It is not Signal PQXDH, and no Signal PQXDH analysis or proof is claimed to transfer to this construction.

7.1 Initiator Flow

1. Verify both endpoint device bundles against expected identities.
2. Extract the responder's verified ML-KEM-768 public key.
3. Encapsulate to obtain a PQ shared secret and PQ ciphertext.
4. Construct transcript material containing the classical Olm session identifier, both device bundles, and the PQ ciphertext.
5. Derive a hybrid session identifier and initial payload key.
6. Encrypt the initial payload with XChaCha20-Poly1305 and context-bound AAD.
7. Serialize the result as HybridEnvelope structural version 1 inside canonical MPQFRAME V2.

7.2 Receiver Flow

8. Parse and validate canonical MPQFRAME V2, the approved cipher suite, HybridEnvelope frame type, and the structural-version-1 payload.
9. Verify expected sender and recipient bundle identities.
10. Decapsulate the ML-KEM ciphertext with the responder private key.
11. Reconstruct the transcript and derive the same session identifier and initial payload key.
12. Authenticate and decrypt the initial payload.
13. Record the dedicated initial-handshake replay identifier only after successful authentication.

7.3 Binding and Security Properties

- The PQ ciphertext is not free-standing; it is included in transcript derivation.
- Both endpoint bundles are included, reducing cross-device and cross-conversation transplantation risk.
- The classical Olm session identifier binds the PQ extension to the established classical context.
- AAD binds the initial payload to its cryptographic context.
- Unsupported PQ algorithms, incorrect classical session identifiers, wrong private keys, substituted bundles, and tampered ciphertext are covered by negative tests.

7.4 Qualification

The module does not implement the complete classical authenticated handshake by itself. It binds ML-KEM material to an existing Olm session context supplied by vodozemac, so the trust boundary includes that classical implementation and the correctness of the session identifier passed into the binding construction. The current profile uses SafeMeet-specific safemeet-core-olm-mlkem-* v2 KDF/AAD domains, a safemeet-olm-mlkem768-v2: session identifier prefix, and MPQFRAME V2 for HybridEnvelope while retaining structural payload version 1.

The core provides dedicated initial Olm-ML-KEM binding replay APIs, and the production wrapper requires replay state and a successful replay commit before plaintext is returned. Lower-level primitives remain available for tests or advanced integrations, so production callers must use the replay-aware high-level path or provide an equivalent at-most-once commit discipline.

8. One-to-One SafeMeet EC-PQ Epoch Ratchet

After setup, SafeMeet uses EcPqEpochRatchetState, the implementation of the SafeMeet EC-PQ Epoch Ratchet. The construction combines a per-message classical EC ratchet branch, a sparse ML-KEM epoch branch, a domain-separated HKDF-SHA-256 final combiner, and XChaCha20-Poly1305 authenticated encryption.

8.1 EC Message Branch

ECMessageRatchetState derives per-message classical material using X25519-based ratchet state. On receive, the next state is represented as a tentative receive step. The branch does not irreversibly advance when the envelope is merely parsed or when key derivation begins.

8.2 Sparse PQ Epoch Branch

GeneralSparsePQState injects ML-KEM entropy at epoch boundaries rather than performing a KEM operation for every message. Each epoch establishes a PQ chain key, from which per-message PQ keys are derived. This amortizes the cost and size of post-quantum operations while periodically refreshing the PQ contribution.

8.3 Per-Message Hybrid Key

For every message, the final AEAD key depends on the EC message key, the PQ message key, the classical session identifier, message number, PQ epoch identifier, epoch message index, sender next EC public key, and any PQ epoch update. AAD independently authenticates the corresponding envelope metadata.

8.4 Bounded State

Limit	Default	Purpose
Pending messages	128	Bounds envelopes waiting for an unavailable PQ epoch.
Skipped message keys	1,024	Bounds out-of-order EC/PQ key material.
Future epoch gap	8	Rejects messages referring to implausibly distant future epochs.

- Duplicate pending envelopes are rejected.
- Message and epoch metadata are cross-checked between EC, PQ, and envelope state.
- Temporary final, EC, and PQ message keys are zeroized after use.

8.5 Staged Decrypt and Commit-after-AEAD

```
parse canonical envelope
  -> check policy, replay, gaps, and duplicates
  -> prepare EC receive step
  -> prepare or resolve PQ receive step
  -> derive final message key and AAD
  -> authenticate/decrypt XChaCha20-Poly1305
  -> on failure: discard tentative state
  -> on success: commit EC state, PQ state, counters, and replay metadata
```

This is an important implementation property. Invalid ciphertext cannot directly advance the EC branch, install a PQ epoch update, or consume the staged message state. The term “transactional” should nevertheless be understood as a cryptographic state-machine property: the core prepares and commits in-memory state in a controlled order. It is not, by itself, a claim about an external persistence transaction.

9. Replay, Ordering, and State-Poisoning Resistance

9.1 Replay Domains

Object	Replay identity / mechanism
Initial Olm-ML-KEM binding envelope	Dedicated domain-separated initial-handshake replay key.
One-to-one EC-PQ epoch envelope	Replay record bound to owner, peer, session, epoch, message number, and epoch index.
Pending envelope	Duplicate pending detection before buffering.
Room message	Seen message indexes plus source and room/session binding.
Room-state event	Unique event_id and linear prev_event_id chain.

The core’s responsibility is to reject repeated cryptographic objects and cross-context reuse. Application-level retry semantics are not part of the cryptographic claim; callers should not interpret replay rejection as a complete idempotent delivery protocol.

9.2 Out-of-Order Delivery

Within configured bounds, skipped EC and PQ keys allow a delayed message to be decrypted after later messages have arrived. If the required PQ epoch is not yet available, the envelope may be buffered. Pending entries and skipped-key storage are bounded to prevent attacker-controlled memory growth.

9.3 State-Poisoning Resistance

State poisoning occurs when malformed input causes a receiver to advance before authenticity is known. SafeMeet mitigates this by separating parsing, tentative key/state derivation, AEAD authentication, and final commit. The same principle is applied to optional PQ updates: a malicious update cannot be installed solely because it appears in a parseable envelope.

10. Canonical Framing and Downgrade Resistance

10.1 MPQFRAME

Field / rule	Security purpose
Magic value	Reject unrelated or legacy byte formats on strict paths.
Protocol version	Reject unsupported wire versions.
Cipher suite	Bind the frame to the approved hybrid primitive suite.
Frame type	Prevent cross-envelope parser confusion.
Reserved field = 0	Reject ambiguous future flags and malformed headers.
Length-prefixed payload	Detect truncation and malformed lengths.
Strict end-of-input	Reject trailing-byte smuggling.

10.2 Protocol Encoding

protocol_encoding.rs provides explicit little-endian integer encoding, length-prefixed bytes and strings, fixed-length reads, UTF-8 validation where relevant, overflow checks, truncation rejection, and ensure_finished() checks. Envelope-specific parsers additionally verify the expected MPQFRAME type and cipher suite.

10.3 Legacy Compatibility

The source still contains from_canonical_or_legacy_bytes() helpers for tests and non-production builds. In the inspected production configuration, those functions are guarded by cfg(any(test, not(feature = "production"))) and legacy-migration cannot be combined with production because lib.rs emits compile_error!. Therefore, the production feature profile does not expose transparent legacy fallback on the strict canonical path.

A small API-hardening improvement would be to gate all legacy helpers exclusively behind a named legacy-migration feature, rather than compiling them for every non-production build. This would reduce accidental use during integration without changing the current production posture.

11. Trust, TOFU, and Verification Evidence

11.1 TOFU Pinning

The trust store pins the Ed25519 identity key and a domain-separated hash of the ML-DSA public identity key for each expected user/device pair. Later observations must match the pinned material. A valid but replaced bundle therefore fails after the first accepted observation.

11.2 Safety Numbers

Safety numbers are derived from ordered local and peer identity material under a dedicated domain. They provide a human-verifiable channel for increasing identity confidence beyond TOFU, for example by comparison in person, by voice, or through a QR workflow implemented by the host application.

11.3 Evidence Model

Verification level	Evidence
TOFU only	Bundle signatures are valid and identity is pinned, but no independent ceremony is supplied.
Safety number	Presented and expected safety numbers match.
Cross-signing	Caller-supplied verification binds the expected subject and bundle fingerprint to a verified signing chain.
Key transparency	Caller-supplied proof result binds subject and fingerprint and reports verified inclusion, consistency, and freshness.

identity_verification.rs evaluates evidence and rejects subject mismatch, fingerprint substitution, stale or incomplete transparency results, and downgrade from stronger existing trust states. Cross-signing and key-transparency verification engines themselves are interfaces, not implementations inside the core.

TOFU limitation
TOFU detects replacement after pinning but does not independently prevent a first-contact man-in-the-middle attack. Applications requiring stronger identity confidence can require safety-number comparison, cross-signing, or key-transparency evidence according to deployment policy.

12. Group Sender-Key Cryptography

12.1 Room Message Encryption

Each sender creates an outbound room session with a random symmetric chain key. For each room message, the session derives a message key and the next chain key, then encrypts with XChaCha20-Poly1305. AAD binds room ID, session ID, sender user ID, sender device ID, and message index.

The inbound room session checks the expected room, session, sender, device, and message-index behavior. It tracks seen indexes and supports bounded out-of-order decryption through skipped room-message keys. State advances only after successful AEAD authentication.

12.2 Room-Key Distribution

RoomKeyPayload is encoded canonically and sealed separately for each eligible device through a one-to-one E2EE channel. Production policy requires that channel to be marked PQ-protected. The receiver validates intended recipient, room, session, sender, and framing before constructing an inbound room session.

12.3 Verified Room State and Epoch Policy

- Trusted room signers are populated only from already verified device bundles and trust decisions.
- Room-state events are ML-DSA signed and form a linear chain through prev_event_id.
- Duplicate event IDs, wrong previous events, invalid signer roles, membership skips, and epoch inconsistencies are rejected.
- Room-key sharing context binds active sender and recipient membership, room epoch, member epoch, and a room-state hash.
- Unsupported room-key requests and forwarded-room-key flows fail closed rather than silently exposing keys.

12.4 Security Qualification

The correct claim is: SafeMeet provides sender-key group encryption whose room keys can be distributed over PQ-protected one-to-one E2EE channels, with verified room-state and epoch policy. It is not a full post-quantum group key agreement protocol. The symmetric sender-key chain also provides weaker post-compromise recovery than the one-to-one EC-PQ epoch ratchet; recovery depends on generating and redistributing a fresh room session after a rotation trigger.

13. Backup-Envelope Cryptography

backup_envelope.rs defines a client-side cryptographic envelope for exporting and restoring typed E2EE records. This section evaluates only the envelope and validation contract, not any backup lifecycle or recovery service.

13.1 Envelope Construction

- Requires a high-entropy recovery key of at least 32 bytes.
- Generates a 32-byte random salt and 24-byte XChaCha nonce.
- Derives an encryption key with HKDF-SHA-256 under a backup-specific domain.
- Encrypts serialized typed records with XChaCha20-Poly1305.
- Authenticates owner user ID, owner device ID, backup ID, backup version, creation time, schema version, salt, and nonce as AAD.

13.2 Parser and Context Validation

The MPQBACKUP envelope contains explicit magic, envelope version, schema version, length-prefixed context strings, version counters, salt, nonce, and length-prefixed ciphertext. The parser rejects bad magic, unsupported versions, malformed lengths, truncation, invalid context, and trailing bytes. The sealing path also rejects records that do not belong to the owner/device context being authenticated.

13.3 Security Properties and Limitations

Negative tests cover wrong recovery keys, wrong user/device context, backup version drift, header tampering, salt and nonce tampering, ciphertext and tag tampering, malformed lengths, truncation, and trailing bytes. This is a meaningful cryptographic primitive, but it should be described as an encrypted backup/export envelope rather than a complete recovery system.

14. Fail-Closed Policy and Production Core API

14.1 Production Policy

Policy axis	Production setting
Legacy frames	Rejected
Post-quantum mode	Required
Expected identity verification	Required
Identity change	Fails closed
Missing PQ prekey	Fails closed; no classical-only downgrade
Room-key channel	Must be PQ-protected

E2eePolicy::production() activates all six controls. Strict parsing wrappers call policy checks before decoding and verify the expected approved cipher suite. The default policy is the production profile.

14.2 Compile-Time Feature Gates

The crate forbids unsafe code and rejects production builds that enable test-support, legacy-migration, kyber-legacy, post-quantum-all, or the non-approved ML-KEM backend. Production also requires the mlkem-pqcrypto backend. These compile-time checks reduce the chance that a permissive migration or test configuration is accidentally shipped as the production core.

14.3 API Discipline

ProductionE2eeContext centralizes strict policy and preferred entry points. For the initial Olm-ML-KEM Session Binding path, the high-level open API structurally requires replay state and commits the replay record before returning plaintext. Remaining API-hardening work is concentrated in persistence atomicity, stateful ratchet testing, and deployment integrations rather than omission of initial-handshake replay checking.

15. Implementation and Testing

15.1 Implementation Properties

- #![forbid(unsafe_code)] at crate level.
- Explicit zeroization or Zeroizing wrappers for secret and intermediate key material.
- Redacted Debug implementations for recovery keys, ciphertext containers, and security-sensitive values.
- Security-event redaction tests to prevent secret leakage into logs.
- Strict canonical encoders and decoders with negative boundary tests.

15.2 Adversarial and State Tests

The test suite includes handshake downgrade tests, identity substitution vectors, parser mutation vectors, cross-context replay, one-to-one and group adversarial flows, room epoch policy, multi-device rotation, backup tampering, concurrency races, security-event redaction, and strict production feature gates. The breadth of negative testing is a major strength because most messaging failures occur in composition and state transitions rather than in the nominal encryption call.

15.3 Fuzz Targets

Fuzz target	Coverage
protocol_frame_fuzz	MPQFRAME plus canonical HybridEnvelope, EcPqEpochEnvelope, RoomKeyPayload, RoomMessageEnvelope, RoomInboundSession, SignedPQPublicKey, DevicePublicBundle, and backup parser/open paths.
backup_envelope_fuzz	Backup-envelope parser, context checks, and cryptographic open behavior.
room_key_envelope_fuzz	Room-key payload/session parsers and production room-key open path under arbitrary decrypted bytes.
verified_room_state_fuzz	Signed room-state event sequences, previous-event links, roles, membership epochs, and state invariants.

HybridEnvelope and EcPqEpochEnvelope are exercised by protocol-frame fuzzing. The repository also includes an initial stateful EC-PQ ratchet fuzz target and randomized/fault-injection integration tests. These artifacts establish a basis for long-trace testing; sustained fuzz campaigns, coverage measurements, corpus management, and preserved run evidence remain future work.

15.4 Additional Testing Directions

- The initial repository focuses on the implementation and automated test suite; additional independent analyses may be developed as the project evolves.
- The repository includes initial Tamarin models for the Olm-ML-KEM Session Binding, the EC-PQ Epoch Ratchet, and their combined one-to-one flow, together with a model-to-Rust correspondence map. These models are analysis artifacts rather than completed proofs: machine-checked proof logs, model-completeness review, and independent protocol analysis are not claimed in this version.
- Dedicated constant-time and side-channel measurements of the integration layer are future implementation-analysis tasks.
- The repository contains parser-oriented fuzz targets; longer continuous fuzzing campaigns and corpus management can extend coverage over time.
- Known-answer vectors can be expanded further to cover additional hybrid derivations and complete state-transition traces.

16. Limitations and Future Work

Area	Current state	Future direction
Hybrid composition and EC-PQ epoch ratchet	Implemented with explicit transcript binding, HKDF-SHA-256 v2 combiner, commit-after-AEAD receive logic, and bounded state.	Extend formal and experimental analysis of forward secrecy, post-compromise behavior, and long state traces.

Area	Current state	Future direction
Replay-aware initial session opening	Production API requires replay state and commits the replay record before plaintext return.	Keep higher-level product integration on the replay-aware API and add end-to-end integration tests.
Stateful fuzzing	Parser/envelope fuzz targets, an initial EC-PQ stateful fuzz target, randomized long-trace tests, and persistence fault-injection tests are included.	Run sustained fuzz campaigns with recorded duration, corpus, coverage, crash triage, and reproducible CI evidence.
Hybrid message-key combiner	HKDF-SHA-256 v2 construction and deterministic known-answer coverage are implemented.	Expand independent cross-check vectors and analysis of hybrid-input robustness.
Protocol invariants	Core invariants are represented in source tests, and initial Tamarin models plus a model-to-Rust map are included.	Execute and refine the models, preserve proof logs, expand compromise/progress queries, and obtain independent review of the model-to-code correspondence.
Legacy helper exposure	Legacy compatibility helpers are excluded from the production feature profile.	Optionally isolate all compatibility helpers behind an explicit migration-only feature.
Group post-compromise recovery	Current design uses sender-key sessions and rotation-based recovery.	Define rotation policy more precisely or investigate stronger post-quantum group-key constructions in later versions.
PQ epoch delivery and active recovery	Sparse PQ epoch updates are supported, but active post-compromise recovery depends on delivery of fresh epoch material.	Specify retransmission/progress behavior and define how selective suppression of PQ epoch updates affects or limits recovery claims.
ML-KEM decapsulation-key lifecycle	Private epoch prekeys are consumed only after successful authenticated receive commit, but retention, expiry, replenishment, and maximum in-flight key policy are not fully specified.	Define explicit retention bounds, expiry/retirement, replenishment, crash recovery, and secure deletion semantics.
Crash-atomic persistence	Receive-side staged commit and atomic-state primitives exist, but durable ratchet/replay integration is not a single proven transaction across every production storage path; outbound send atomicity is also incomplete.	Complete durable receive integration and staged/idempotent outbound persistence, then exercise restart and fault-injection tests.
Classical Olm dependency boundary	The initial hybrid binding depends on an already-established Olm session supplied by vodozemac.	Pin and document the exact vodozemac source/revision, local patches, session-ID semantics, reachable protocol profiles, and relevant edge-case behavior.

Area	Current state	Future direction
Local journal authenticity	Replay/state journals use unkeyed SHA-256 checksums for corruption detection.	Use authenticated secret storage and a trusted monotonic source when resistance to a malicious local writer or rollback is required.
Room-key transport evidence	Room-key policy requires PQ-protected one-to-one transport, while the current integration surface relies on the concrete transport reporting this property.	Strengthen the integration with a verifiable capability/evidence object or a concrete transport type that cannot represent a classical-only path.

17. Technical Summary

The following table summarizes which major E2EE components are implemented in the initial project version and notes the main technical qualifications of each component.

Core area	Implementation status	Technical note
Primitives and hybrid derivation	Implemented	ML-KEM-768, ML-DSA-44, Ed25519, X25519, XChaCha20-Poly1305, and domain-separated HKDF-SHA-256 are composed in the SafeMeet protocol profile.
Olm-ML-KEM Session Binding	Implemented	SafeMeet-specific transcript/AAD binding extends an existing Olm context; the production opening path is replay-aware.
One-to-one EC-PQ Epoch Ratchet	Implemented	Per-message EC evolution and sparse PQ epochs are combined with commit-after-AEAD receive semantics and bounded pending/skipped state.
Replay protection	Implemented	Dedicated replay domains and rich context binding cover initial binding, one-to-one messages, pending duplicates, room messages, and room-state events.
Framing and parser hardening	Implemented	Canonical MPQFRAME parsing checks version, cipher suite, frame type, reserved fields, lengths, and trailing bytes.
Identity and bundles	Implemented	Nested Ed25519 + ML-DSA signatures and expected-identity verification bind device and algorithm context.
Trust and verification	Implemented with external evidence hooks	TOFU and safety numbers are implemented; cross-signing and key-transparency verification are integrated as evidence interfaces.
Group cryptography	Implemented with defined scope	Sender-key room encryption and verified room-state policy are implemented; this version is not a full PQ group key-agreement protocol.
Backup envelope	Implemented cryptographic envelope	Typed E2EE records can be exported/imported under HKDF/XChaCha20-Poly1305; recovery service and UX are outside the core.

Core area	Implementation status	Technical note
Fail-closed policy	Implemented	Strict policy axes and compile-time feature guards limit legacy, missing-PQ, identity-change, and non-PQ room-key paths.
Fuzzing and adversarial testing	Implemented baseline; extensible	Parser/envelope fuzzing, an initial stateful EC-PQ fuzz target, randomized long-trace tests, and persistence fault-injection tests are present; sustained campaign evidence remains to be produced.
Implementation quality	Implemented baseline	The crate forbids unsafe code and includes zeroization, redaction, strict parsing, and explicit module boundaries.

Overall implementation
SafeMeet E2EE Core v1.0 provides a working hybrid classical/post-quantum reference implementation covering device identity, Olm-ML-KEM session binding, the EC-PQ epoch ratchet, replay and ordering controls, canonical framing, trust policy, sender-key rooms, and encrypted backup records. The limitations documented in Section 16 define the scope of this initial project version and directions for later development.

18. Conclusion

The first SafeMeet E2EE core version combines post-quantum key establishment and authentication with classical cryptography, applies explicit context binding, and treats hostile asynchronous delivery as a state-machine problem. The nested device-bundle signatures, SafeMeet Olm-ML-KEM Session Binding, canonical MPQFRAME parser, bounded SafeMeet EC-PQ Epoch Ratchet, replay domains, verified room state, and fail-closed policy form a coherent cryptographic architecture. The SafeMeet-specific names identify the project's own constructions and do not imply protocol identity with similarly motivated systems.

A central design property is commit-after-AEAD. By staging EC and PQ receive steps and refusing to commit them before ciphertext authenticity succeeds, the design reduces the risk of attacker-induced state desynchronization. Negative tests, parser fuzzing, concurrency tests, and explicit feature gates provide additional implementation coverage.

Future technical work includes sustained stateful-fuzz campaigns, broader deterministic vectors, deeper analysis of hybrid composition and post-compromise behavior, stronger persistence integration, explicit sparse-PQ epoch progress rules under selective message loss, and refinement of ML-KEM epoch-key retention, expiry, replenishment, and deletion rules. Group messaging in this version remains a sender-key construction with PQ-protected pairwise room-key distribution and rotation-based recovery rather than a full post-quantum group key-agreement protocol.

This whitepaper and the accompanying Rust source code define the initial SafeMeet E2EE project version. They provide a concrete reference for the project's algorithms, message formats, state transitions, implementation boundaries, and current technical limitations, and establish the baseline from which later protocol and implementation versions can evolve.

Reference implementation snapshot. This whitepaper corresponds to safe-e2ee-core-production-v1.0.zip (SHA-256 c88928d64b07cd0789481b4f21ba9a7fe405df20dc8589158853bd75dc383210). The exact sibling vodozemac dependency used for reproduction is distributed separately in the v1.0 release package; its archive SHA-256 is 4c19093a137f5b794f56e9e4db4e1ef6b2818f41e5c751940fa23d6575c146d9.

Appendix A. Core Module Map and Source Paths

Module	Source path	Core security role
protocol_frame.rs	src/protocol/protocol_frame.rs	Canonical MPQFRAME format, version, suite, type, reserved-field, length, and trailing-byte validation.
protocol_encoding.rs	src/protocol/protocol_encoding.rs	Strict length-prefixed encoding and decoding utilities.
pq.rs	src/crypto/pq.rs	ML-KEM-768 key generation, encapsulation, and decapsulation wrapper.
mldsa_identity.rs	src/crypto/mldsa_identity.rs	ML-DSA-44 identity signing and verification.
signed_pq_key.rs	src/identity/signed_pq_key.rs	Dual-signed ML-KEM public key and algorithm/device binding.
device_bundle.rs	src/identity/device_bundle.rs	Dual-signed device bundle and expected identity verification.
hybrid_olm_mlkem.rs	src/session/hybrid_olm_mlkem.rs	Hybrid classical/PQ session setup, transcript derivation, and initial payload protection.
hybrid_envelope.rs	src/protocol/hybrid_envelope.rs	Canonical initial setup envelope.
hybrid_aead.rs	src/crypto/hybrid_aead.rs	XChaCha20-Poly1305 sealing/opening and AAD handling.
ec_message_ratchet.rs	src/ratchet/ec_message_ratchet.rs	Per-message EC ratchet with tentative receive steps.
general_sparse_pq.rs	src/ratchet/general_sparse_pq.rs	Sparse ML-KEM epoch ratchet and skipped-key handling.
ec_pq_epoch_ratchet.rs	src/ratchet/ec_pq_epoch_ratchet.rs	Hybrid one-to-one ratchet, bounds, pending buffering, staged decrypt, and commit.
ec_pq_epoch_envelope.rs	src/protocol/ec_pq_epoch_envelope.rs	Canonical one-to-one message envelope.
replay_protection.rs	src/state/replay_protection.rs	Initial and message replay identifiers and replay state.
e2ee_state_transaction.rs	src/state/e2ee_state_transaction.rs	Transactional receive-state staging and atomic persistent state batches.
trust_store.rs	src/trust/trust_store.rs	TOFU pinning, identity-change detection, safety-number support, and device trust state.
identity_verification.rs	src/identity/identity_verification.rs	Safety-number, cross-signing, and key-transparency evidence evaluation.

room_session.rs	src/group/room_session.rs	Sender-key room encryption, skipped room keys, and seen-index checks.
room_key_distribution.rs	src/group/room_key_distribution.rs	Canonical room-key payload and per-device one-to-one sealing/opening.
verified_room_state.rs	src/trust/verified_room_state.rs	ML-DSA-signed room-state chain, roles, membership, epochs, and state hash.
room_group_policy.rs	src/group/room_group_policy.rs	Room-key sharing eligibility, epoch policy, withholding, and unsupported-flow rejection.
backup_envelope.rs	src/backup/backup_envelope.rs	HKDF/XChaCha20-Poly1305 encrypted backup/export envelope.
e2ee_policy.rs	src/runtime/e2ee_policy.rs	Six-axis fail-closed production policy.
production.rs	src/runtime/production.rs	Strict high-level E2EE context and integration entry points.
e2ee_security_event.rs	src/runtime/e2ee_security_event.rs	Security-event classification and redaction behavior.
lib.rs	src/lib.rs	Unsafe-code prohibition and compile-time feature incompatibility gates.

Appendix B. Terminology and Concepts

This appendix provides a consolidated glossary of the principal cryptographic, protocol, state-management, trust, group-messaging, backup, and implementation terms used in this whitepaper. Definitions are specific to the SafeMeet Core E2EE v1.0 context and should be read together with the detailed descriptions in the main body.

No.	Term / Concept	Abbreviation / Related name	Definition in this paper
1	**End-to-End Encryption**	E2EE	Encryption in which ordinary message plaintext is protected at the sending endpoint and recovered only at the intended receiving endpoint. Intermediate servers are not expected to have access to plaintext.
2	**Post-Quantum Cryptography**	PQC	Cryptographic algorithms designed to remain secure against attacks from sufficiently capable quantum computers.
3	**Hybrid Classical–Post-Quantum Cryptography**	Hybrid Cryptography	A construction that combines classical and post-quantum cryptographic inputs so that security does not rely exclusively on a single cryptographic family.
4	**ML-KEM-768**	Module-Lattice-Based Key Encapsulation Mechanism	A post-quantum key encapsulation mechanism used by SafeMeet to derive shared secret material during initial session setup and later PQ epochs.
5	**ML-DSA-44**	Module-Lattice-Based Digital Signature Algorithm	A post-quantum digital signature scheme used to authenticate device identity and device-bundle key material.
6	**Ed25519**	Ed25519 Digital Signature	A classical elliptic-curve digital signature algorithm used as part of SafeMeet device authentication.
7	**X25519**	X25519 Key Agreement	A classical elliptic-curve key-agreement mechanism used in the EC branch of the one-to-one ratchet.
8	**XChaCha20-Poly1305**	AEAD Cipher	An authenticated-encryption algorithm used to protect message payload confidentiality and integrity.
9	**Authenticated Encryption with Associated Data**	AEAD	Encryption that protects plaintext confidentiality while also authenticating the ciphertext and associated metadata.
10	**Additional Authenticated Data**	AAD	Metadata authenticated by AEAD but not encrypted. It is used to bind ciphertext to protocol and message context.
11	**Shared Secret**	Shared Secret Material	Secret material jointly derived or established by two endpoints and subsequently used for key derivation.
12	**Key Material**	-	A general term for cryptographic values such as private keys, shared secrets, chain keys, message keys, and intermediate secrets.
13	**Message Key**	Per-Message Key	A cryptographic key derived for a particular message and normally not reused for unrelated messages.
14	**Chain Key**	-	A stateful key that evolves over time and is used to derive subsequent message keys.
15	**Cryptographic Ratchet**	Ratchet	A mechanism that evolves cryptographic state and keys over time, limiting the effect of compromise of a particular state.
16	**SafeMeet EC-PQ Epoch Ratchet**	EC-PQ Epoch Ratchet	SafeMeet's one-to-one ratchet combining a classical EC message-ratchet branch with a sparse post-quantum epoch branch.
17	**EC Ratchet Branch**	EC Branch	The classical branch of the SafeMeet ratchet, based on elliptic-curve key-agreement material.
18	**Sparse PQ Branch**	PQ Branch	The post-quantum branch in which ML-KEM updates occur at selected epochs rather than for every individual message.
19	**Epoch**	PQ Epoch	A PQ interval in which an ML-KEM update establishes an epoch chain key from which per-message PQ key material is derived.
20	**Sparse PQ Epoch**	Sparse Post-Quantum Epoch	A PQ update interval activated periodically rather than on every message, reducing computational and bandwidth cost.
21	**Hybrid Session Setup**	-	Session establishment that combines classical session context with post-quantum ML-KEM shared secret material.
22	**SafeMeet Olm-ML-KEM Session Binding**	-	SafeMeet's hybrid setup mechanism that combines verified ML-KEM material with an existing classical Olm session context.

No.	Term / Concept	Abbreviation / Related name	Definition in this paper
23	**Olm Session**	Olm	The classical session context used as one component of SafeMeet's hybrid session establishment.
24	**Device Identity**	-	Cryptographic identity material used to associate public keys with a particular user and device.
25	**SignedPQPublicKey**	-	A SafeMeet structure that binds an ML-KEM public key to user, device, and algorithm context using classical and post-quantum signatures.
26	**DevicePublicBundle**	Device Bundle	A signed collection of public device information and cryptographic keys used to establish and verify a device identity.
27	**Identity Binding**	-	The process of cryptographically associating public keys with the expected user, device, and algorithm context.
28	**Identity Substitution Resistance**	-	The property that prevents a valid key or bundle belonging to one subject from being silently accepted as belonging to another.
29	**Trust On First Use**	TOFU	A trust model in which the first accepted identity is pinned and later unexpected identity changes fail closed or require explicit handling.
30	**Identity Pinning**	-	Storing an accepted identity or fingerprint so that later changes can be detected.
31	**Safety Number**	-	A human-comparable fingerprint derived from local and peer identity material and used to assist manual identity verification.
32	**Trust Model**	-	The set of rules determining when a key, device, identity, or verification result is considered trusted.
33	**Message Envelope**	Envelope	A structured representation containing ciphertext and authenticated metadata required for protocol processing.
34	**MPQFRAME**	Canonical MPQFRAME	SafeMeet's canonical binary framing format that binds protocol version, cipher suite, frame type, lengths, and reserved-field rules.
35	**Canonical Encoding**	Canonical Framing	An encoding rule under which valid data has one well-defined representation, reducing parser ambiguity and alternative encodings.
36	**Cipher Suite**	-	A defined combination of cryptographic algorithms and protocol parameters used together.
37	**Strict Parsing**	Strict Decoding	Parsing behavior that rejects malformed, non-canonical, unsupported, truncated, oversized, or trailing data rather than attempting permissive recovery.
38	**Reserved Field**	-	A protocol field reserved for future use and required to have a defined value, typically zero, in the current protocol version.
39	**Replay Attack**	Replay	An attack in which a previously valid ciphertext or envelope is delivered again to cause repeated processing.
40	**Replay Protection**	-	Logic that detects and rejects previously processed messages or envelopes.
41	**Message Reordering**	Reordering	Delivery of messages in an order different from the order in which they were sent.
42	**Skipped Message Key**	Skipped Key	A temporarily retained key that allows a bounded number of out-of-order messages to be decrypted later.
43	**Bounded Skipped Keys**	-	A fixed limit on retained skipped-message keys to prevent unbounded memory growth.
44	**Pending Queue**	Pending Message Queue	A bounded queue containing messages that cannot yet be processed because the required state is not available.
45	**Maximum Future-Epoch Gap**	Future-Epoch Gap	A limit on how far ahead of the current state a future-epoch message may be accepted for deferred processing.
46	**State Poisoning**	State-Poisoning Attack	An attack in which invalid input attempts to advance or corrupt long-lived cryptographic state so that later legitimate messages fail.
47	**Commit-after-AEAD**	Commit-after-AEAD Discipline	A receive-side rule in which candidate state is prepared first, ciphertext is authenticated, and the state is committed only after successful AEAD verification.

No.	Term / Concept	Abbreviation / Related name	Definition in this paper
48	**Tentative State**	Prepared State	Cryptographic state computed provisionally during receive processing but not yet applied to the long-lived session state.
49	**State Commit**	Cryptographic State Commit	The operation that makes a previously prepared ratchet state permanent after successful message authentication.
50	**Fail-Closed Policy**	Fail Closed	A security policy under which unexpected, unsupported, incomplete, or invalid cryptographic conditions cause rejection rather than fallback to a weaker mode.
51	**Downgrade Attack**	-	An attack that attempts to force participants to use an older or weaker protocol version, algorithm, or security configuration.
52	**Downgrade Resistance**	-	The ability to reject unsupported or weaker configurations instead of silently falling back to them.
53	**Production Feature Gate**	Feature Gate	Compile-time or build-time controls that determine which protocol and compatibility features are permitted in production builds.
54	**Legacy Parser**	Legacy Parsing Path	A parsing path supporting an older protocol representation, generally retained for migration or compatibility rather than normal production processing.
55	**Adversarial Delivery Path**	-	A threat-model assumption in which the message-delivery path may observe, delay, duplicate, reorder, drop, or inject data.
56	**Sender Key**	-	Cryptographic key material maintained by a group sender and used to encrypt a sequence of room messages.
57	**Room Session**	Sender-Key Room Session	The sender-key chain and associated state used by one sender to encrypt messages in a group room.
58	**Room Key**	-	Cryptographic key material used by the group sender-key construction and distributed to authorized room members.
59	**Room Epoch**	-	A version or generation of the room's cryptographic state used to bind group-message processing to a particular room-key state.
60	**Member Epoch**	-	A version of member state used to bind group cryptographic processing to the expected membership state.
61	**Room-Key Rotation**	Key Rotation	Replacement of an existing room key with fresh cryptographic material, for example after membership or security-state changes.
62	**Post-Compromise Recovery**	PCR / Post-Compromise Security	The ability of a protocol to restore confidentiality or security for future messages after a previous state or key has been compromised.
63	**Group Post-Compromise Recovery**	Group PCR	Recovery of group-message security following compromise of room or sender-key material; in the current SafeMeet design this relies mainly on room-key rotation.
64	**Backup Envelope**	-	An encrypted and authenticated structure used to store or transport typed E2EE records for backup and recovery.
65	**Recovery Key**	-	A secret key used to encrypt and decrypt protected E2EE backup material.
66	**High-Entropy Recovery Key**	-	A recovery key generated with high cryptographic randomness and a sufficiently large key space to resist guessing and brute-force attacks.
67	**Typed E2EE Record**	-	An E2EE backup record whose data type is explicitly identified and authenticated to prevent interpretation under an unintended record type.
68	**Zeroization**	-	Secure clearing or overwriting of sensitive cryptographic values in memory when they are no longer needed.
69	**Redaction**	-	Removal or masking of sensitive information from logs, diagnostics, telemetry, error messages, or other observable output.
70	**Fuzz Testing**	Fuzzing	Automated testing that feeds malformed, unexpected, or pseudo-random inputs into parsers or APIs to identify crashes and security-relevant edge cases.
71	**Fuzz Target**	-	A specific parser, function, or API selected as the input target for fuzz testing.
72	**Reference Implementation**	-	The source-code implementation used to demonstrate and document how the protocol design is realized in software.

No.	Term / Concept	Abbreviation / Related name	Definition in this paper
73	**Implementation-Grounded Description**	Implementation-Grounded	A protocol description based directly on the behavior and structure of the inspected source implementation rather than solely on an abstract design.
74	**Formal Security Proof**	Formal Proof	A mathematical or machine-assisted proof that specified security properties hold under explicit assumptions. This whitepaper does not claim to provide such a proof.

End of SafeMeet Core E2EE Security Whitepaper v1.0