SkyBridge Compass: A Crypto-Agile P2P Handshake Framework for Remote Desktop and File Transfer with Auditable Downgrade Policy and Post-Quantum Readiness

Zi'ang Li

Abstract—Peer-to-peer remote desktop and file transfer systems increasingly operate across heterogeneous devices, untrusted networks, and long lifecycles in which cryptographic assumptions may expire faster than products. Yet "post-quantum readiness" is often treated as a one-off algorithm swap rather than a migration problem with downgrade resistance, auditability, and reproducibility requirements. This paper presents SkyBridge Compass, a crypto-agile P2P system that explicitly separates (i) negotiated cryptographic suites, (ii) provider backends, and (iii) a policy layer that governs when (and whether) classic fallback is permissible. We use an actor-isolated handshake state machine with deterministic outcomes and structured security-event emission, enabling end-to-end audit trails for downgrade, identity mismatch, and signature verification failures. We evaluate three configurations on macOS 26.x as a deployment instance: classic (X25519 + Ed25519), PQC via liboqs (ML-KEM-768 + ML-DSA-65), and PQC via native CryptoKit (ML-KEM-768 + ML-DSA-65). We do not claim perfect forward secrecy for the KEM-based PQC suites in v1; post-quantum confidentiality holds only while long-term KEM private keys remain uncompromised. End-to-end handshake completion latency (including event emission overhead) is 1.62 ms (p95 1.81 ms) for classic, 2.29 ms (p95 3.01 ms) for liboqs PQC, and 5.76 ms (p95 6.71 ms) for CryptoKit PQC over N=1000 iterations. The handshake wire size (MessageA + MessageB + 2×Finished) grows from 827 B (classic) to 12,163 B (PQC); X-Wing hybrid KEM measures 12,195 B. Beyond performance, we validate downgrade policies, failure semantics, and audit-signal fidelity under fault injection, and we release scripts and artifacts that reproduce all primary figures and tables.

Index Terms—post-quantum cryptography, crypto agility, downgrade resistance, secure handshake, peer-to-peer systems, remote desktop, auditability, reproducibility, platform deployment.

✦

1 INTRODUCTION

Peer-to-peer (P2P) remote desktop and file transfer systems must establish secure sessions under adversarial network conditions, device churn, and multi-year lifecycles. These systems face a practical tension: users demand "it just works" pairing across heterogeneous stacks, while defenders require that cryptographic negotiation remains robust against downgrade and implementation-level failure modes. Post-quantum cryptography (PQC) increases this tension. Larger messages, higher computational cost, and uneven platform support make fallback tempting—yet fallback is exactly where silent downgrade and unverifiable behavior tend to hide.

Existing secure-channel designs and frameworks provide mature cryptographic building blocks, but they do not, by default, solve the migration problem an implementer actually faces: (i) selecting among classic/PQC/hybrid suites across platform strata, (ii) ensuring that policy (e.g., "require PQC") is enforced at the entry points where callers cannot accidentally violate it, (iii) targeting deterministic failure semantics under timeouts, reorder/duplication, and malformed inputs, and (iv) emitting security-relevant telemetry that can be audited without turning the system into a logging minefield. In practice, many systems either hard-code a single suite, or allow "compatibility" to creep into negotiation in ways that are hard to reason about and harder to verify.

This paper addresses these gaps with SkyBridge Compass, a crypto-agile P2P system designed for explicit PQC migration and auditability, with Apple platforms as a deployment instance. The core idea is to treat crypto agility as a layered contract: negotiated suites define the protocol surface; provider backends implement suites using platform-native or portable cryptography; and a policy layer governs fallback and produces structured evidence when (and only when) downgrade occurs. We complement this design with an actor-isolated handshake driver that enforces one-shot completion, timeouts, and sensitive-material zeroization, yielding deterministic outcomes under the modeled transport faults.

Security Guarantees (Checklist). To reduce ambiguity in reviewer-facing claims, we state the v1 guarantees as a checklist of *verifiable* properties under the threat model in Section 5:

- **Negotiation integrity:** the Responder selects only a suite offered by the Initiator and with a matching key share; unknown suite IDs are rejected if they appear in `keyShares[]` or as `selectedSuite`. *Evidence:* Fig. 1; Table 3; `Handshake DriverTests`.
- **Transcript binding:** `sigB` commits to MessageA and the chosen suite (and policy), so any tampering of `supportedSuites[]`, `keyShares[]`, or `policy` fails verification. *Evidence:* Table 3; `Transcript IntegrityPropertyTests`; `ProtocolSignature RegressionTests`.
- **Downgrade resistance (policy-defined):** `strictPQC` forbids all fallback; the default policy permits fallback only for a local-only whitelist of capability/provider failures (timeouts and unauthenticated network failures cannot trigger

- *Independent Researcher, Tianjin, China. E-mail: 2403871950@qq.com.*

fallback). *Evidence:* Fig. 4; Fig. 5; `PolicyDowngrade
BenchTests`; Table 6.
- **Explicit key confirmation:** two Finished frames provide mutual confirmation, preventing half-open sessions (Fig. 1). *Evidence:* Table 5; `HandshakeFaultInjection
BenchTests`.
- **Auditability of migration decisions:** every fallback emits a structured `cryptoDowngrade` event with a minimal evidence tuple (Section 5), enabling post-hoc verification and detection (e.g., Fig. 7). *Evidence:* Fig. 7; Table 3.
- **Non-claims (scope control):** v1 does *not* claim PQC PFS for KEM-to-static-recipient suites, nor tamper-evident audit logs; see Section 8.

1.1 Contributions

Contributions:

1) We turn PQC migration from a narrative into an enforceable contract by making downgrade and legacy-compatibility decisions explicit, testable, and auditable rather than implicit "best-effort" fallback. *Evidence:* Fig. 5; Table 3; `PolicyDowngradeBenchTests`, `LegacyFallbackPreconditionTests`.

2) We design a crypto-agile P2P handshake that cleanly separates suite negotiation, transcript binding, and protocol-signature keys, preventing "suite says PQC but keys stay classical" class of integration bugs. *Evidence:* Fig. 1; Table 3; `ProtocolSignatureRegressionTests`, `TranscriptIntegrityPropertyTests`.

3) We provide migration safety with measurable coverage by enumerating legacy/PQC strata and validating acceptance/rejection boundaries under fault injection instead of relying on informal compatibility claims. *Evidence:* Table 6; Table 5; `HandshakeFaultInjectionBenchTests`.

4) We harden correctness under concurrency by an actor-isolated state machine with idempotent transitions and bounded retries, eliminating double-resume and state desynchronization failure modes. *Evidence:* Fig. 3; Table 3; `HandshakeDriverTests`.

5) We make evaluation reproducible by shipping a benchmark harness that reports distributional statistics (p50/p95/p99) over large-N runs and links each metric back to a concrete failure taxonomy. *Evidence:* Fig. 4; Table 8; `HandshakeBenchmarkTests`, `MessageSizeSnapshotTests`.

1.2 Paper Organization

Section 2 reviews related work on P2P pairing, negotiation, and PQC migration. Section 3 details system architecture and protocol design. Section 4 presents the handshake state machine. Section 5 presents the security model and guarantees. Section 6 describes implementation highlights and platform instantiation. Section 7 outlines evaluation methodology and metrics. Section 8 discusses limitations and future work, and Section 9 concludes. Supplementary Materials provide full wire-format definitions, implementation details, and extended tables.

2 RELATED WORK

We use the following terms throughout: *crypto agility* (ability to migrate primitives without redesign), *downgrade resistance* (prevent or detect forced fallback), *suite negotiation* (explicit, authenticated selection of handshake KEM/SIG/AEAD/KDF), *PQC readiness* (ability to deploy PQC alongside classic suites), and *auditability* (deterministic, minimal event traces for security decisions). Our threat model assumes an active network attacker attempting to induce fallback or obscure failures; therefore we focus on mechanisms that make negotiation and failures explicit and auditable.

2.1 Peer-to-Peer Device Pairing

Pairing protocols such as Bluetooth SSP [1], Continuity [2], and PAKEs like SPAKE2+ [3] establish initial trust but stop short of post-pairing migration, downgrade policy, or auditability. SkyBridge Compass begins *after* pairing and treats pinned identity keys as inputs rather than outcomes.

2.2 Cryptographic Agility and Negotiation

TLS 1.3 and QUIC bind negotiated parameters into transcripts and transport parameters to prevent silent downgrade [4], [5], but they do not define an auditable, system-level fallback policy across heterogeneous providers. We operationalize migration guidance [6], [7] with explicit policy gates and event emission that make downgrade decisions observable and testable.

2.3 Handshake Frameworks and Noise-Style Patterns

Noise provides a compact framework for describing authenticated key exchange patterns with explicit transcript binding and identity key usage [8]. While SkyBridge Compass does not implement Noise directly, our MessageA/MessageB/Finished exchange targets similar transcript-binding goals while adding explicit suite negotiation and auditable policy outcomes.

2.4 Post-Quantum Cryptography Deployment

PQC standardization and early deployments highlight size and performance costs, along with pressure for hybrid deployments [9], [10], [11], [12], [13], [14], [15]. We focus on making downgrade policy and auditability explicit across providers, treating Apple's CryptoKit PQC [16] as a deployment instance rather than a protocol dependency.

2.5 Secure State Machine Design

State machine vulnerabilities have been a persistent source of security bugs in protocol implementations [17], [18]. Actor-based concurrency models, as implemented in Swift's actor system [19], provide compile-time guarantees against data races. We extend this model with deterministic completion, timeout handling, and audit events for cryptographic handshakes.

2.6 Baseline Comparison with Widely Deployed Handshakes

Table 1 contrasts SkyBridge Compass with widely deployed handshake frameworks on explicit negotiation, downgrade auditability, and failure semantics, and reports message count, wire size, and latency where available.

TABLE 1
Baseline comparison with widely deployed handshakes. Message counts and negotiation semantics are summarized from the TLS/QUIC/DTLS RFCs and Noise specification [4], [5], [8], [20]. Loopback latencies and wire sizes are measured with BaselineBenchRunner (release), N=1000; wire sizes report loopback p50/p95 (kB) and are cert-dependent for TLS/QUIC/DTLS. Noise wire size is pattern-dependent and reflects the XX pattern used in this baseline.

Protocol	Suite nego. explicit	Auditable downgrade policy	Failure semantics	Msgs	Wire size (p50/p95, kB)	p50/p95 latency
TLS 1.3 [4]	Yes (cipher suites, groups)	Downgrade prevention only; audit external	Alerts defined; audit external	4 (1 RTT)	8.6/8.7	8.53/14.01 ms
QUIC [5]	Yes (TLS 1.3)	Same as TLS 1.3	Same as TLS 1.3	4 (1 RTT)	7.6/14.1	0.34/11.37 ms
WebRTC DTLS [20]	Yes (DTLS 1.2 ciphers)	Downgrade prevention only; audit external	Alerts defined; audit external	4 (1 RTT)	3.0/3.1	8.18/11.79 ms
Noise XX [8]	No (fixed pattern)	No policy layer	Pattern-defined; audit external	3	2.5/2.6	0.44/0.58 ms
					4.8/4.9 (classic)	1.44/1.57 ms (classic)
					36.9/43.6 (liboqs)	1.95/2.76 ms (liboqs)
SkyBridge Compass	Yes (suite + policy bound)	Yes (explicit policy + events)	Deterministic + evented	4	18.6/18.8 (CryptoKit)	5.91/7.53 ms (CryptoKit)

Loopback SkyBridge p95 latency is 1.57/2.76/7.53 ms for classic/liboqs/CryptoKit in our loopback harness; these numbers are not directly comparable to TLS/QUIC/DTLS handshakes in production deployments, but provide a reproducible baseline under a consistent measurement setup. The in-memory handshake microbench isolates crypto/state-machine overhead (p95 1.45/2.62/5.45 ms for classic/liboqs/CryptoKit) and is reported in Section 7 and Supplementary Tables S1–S2. Loopback TLS/QUIC/DTLS p95 latency is 11.37–14.01 ms; wire-size baselines (p50/p95) are reported in Supplementary Table S7 and are reproducible via loopback pcap capture (`Scripts/run_baseline_bench.sh`). SkyBridge-CryptoKit loopback baselines are included in Supplementary Table S7.

3 SYSTEM ARCHITECTURE

3.1 Overview

SkyBridge Compass implements a layered architecture separating concerns across four primary domains: discovery, cryptographic operations, handshake management, and session transport. Fig. 2 illustrates the high-level component relationships and trust boundaries.

3.2 Threat Model

We model an active network attacker in the Dolev-Yao style with the following **capability set**:

Network Capabilities (Attacker-Controlled):

- **Drop / delay / reorder / duplicate** packets on the discovery channel
- **Modify/inject** arbitrary bytes in MessageA, MessageB, or Finished frames
- **Replay** prior handshake messages across sessions
- **Spoof capability or policy claims** to induce weaker negotiation
- **Force negotiation failure** by corrupting suites, key shares, or signatures

Defense Mapping (Protocol + Policy):

- **Transcript binding:** `sigB` covers MessageA and the chosen suite, so modifications to `supportedSuites[]`, `keyShares[]`, or `policy` fail verification.

- **Negotiation integrity:** Responder MUST select a suite that Initiator offered and for which a key share exists; otherwise reject `missingKeyShare`.
- **Replay control:** `handshakeId` is derived from nonces and cached to reject duplicates within a window.
- **Downgrade resistance:** timeout-triggered fallback is disallowed; only whitelisted errors may fallback under default policy; strictPQC forbids all fallback.
- **Rate limiting:** per-peer fallback cooldown prevents rapid downgrade cycling.
- **Legacy gating:** legacy P-256 acceptance requires an authenticated channel or an existing trust record.

Trust Assumptions:

1) The initial pairing ceremony occurs over a trusted out-of-band channel (e.g., visual PIN comparison on both device screens)
2) The device Keychain provides integrity guarantees for stored identity keys
3) The Secure Enclave (when available) provides hardware-backed key isolation that resists software-level extraction
4) Users can visually verify device identity during pairing (no blind trust)

Out of Scope:

- Side-channel attacks on cryptographic implementations
- Physical attacks on Secure Enclave hardware
- Compromise of the operating system kernel
- Social engineering attacks on users

3.3 Protocol Message Flow

Fig. 1 illustrates the handshake message sequence and transcript coverage.

MessageA carries the `supportedSuites[]` (offered suites) and `keyShares[]` arrays (up to two entries to bound size); MessageB selects a suite and binds MessageA via `sigB`. Both sides verify that the selected suite was offered and that a matching key share exists, preventing negotiation mismatch. Under the two-attempt strategy (Section 6.7), PQC and classic are not co-offered in a single `MessageA`; when fallback is permitted, the Initiator retries with a classic-only `MessageA` after a whitelist of locally generated capability and negotiation failures. Explicit
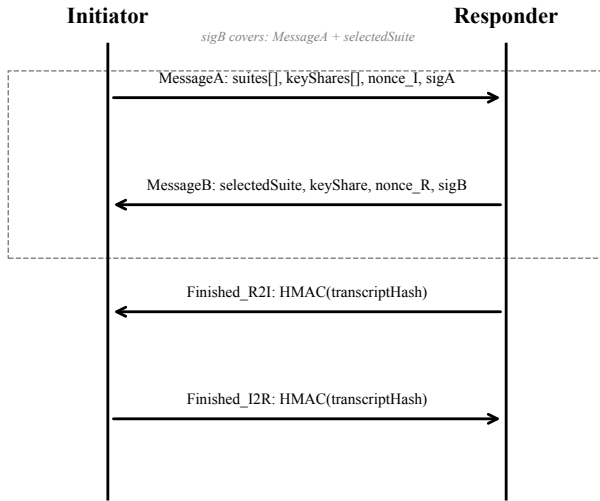
Fig. 1. Handshake sequence with transcript coverage and policy visibility.

key confirmation uses two small Finished frames to avoid half-open state. Full wire-format definitions, field validation rules, and encoding rationale appear in the Supplementary Materials (Wire Format and Validation Rules).

Failure Semantics:

- All failures trigger `HandshakeContext.`
  `zeroize()` before error propagation
- Security events are emitted for audit logging
- No partial state is retained after failure

3.4 CryptoProvider Protocol

The CryptoProvider protocol defines a unified interface for all cryptographic operations required by the handshake and session layers [21]. This abstraction enables transparent substitution of underlying implementations without modifying caller code.

The `SigningKeyHandle` abstraction resolves the apparent conflict between "private key as Data" and "Secure Enclave keys never leave hardware." Software keys are one variant; hardware-backed keys use a reference or callback, ensuring the protocol interface does not assume exportability.

The protocol mandates `Sendable` conformance, ensuring thread-safe usage across Swift's structured concurrency model. Each provider exposes its tier classification (`nativePQC`, `liboqsPQC`, or `classic`) and active cipher suite, enabling runtime introspection for logging and policy enforcement.

3.5 CryptoSuite Wire Format

A **suite** defines the handshake tuple (`KEM, SIG, AEAD, KDF`) used for the KEM-DEM envelope and transcript binding. Data-plane AEAD is negotiated separately and is fixed to AES-256-GCM in v1 (Table 10). Suite identifiers use a compact 16-bit wire ID with reserved ranges for hybrid, PQC, classic, and experimental suites to enable forward compatibility. The full range table, encoding rules, and X-Wing notes appear in the Supplementary Materials (Suite Identifiers and Wire Format). Unknown suite identifiers are parsed as `unknown(wireId)`

for audit visibility and forward compatibility; negotiation rejects unknown values if they appear in `keyShares` or as `selectedSuite`.

3.6 Provider Factory and Selection

The `CryptoProviderFactory` implements deterministic provider selection based on runtime capability detection and policy configuration:

The selection algorithm proceeds as follows:

1) **Capability Detection:** Query the environment for native PQC availability and liboqs presence.
2) **Policy Application:** Match detected capabilities against the requested policy.
3) **Provider Instantiation:** Create the appropriate provider instance for the detected environment and policy.
4) **Event Emission:** Emit a `SecurityEvent` recording the selection decision.

For `preferPQC` policy, the precedence order is `ApplePQCProvider → OQSPQCProvider → ClassicProvider`. This order is deterministic. The `requirePQC` policy returns an `UnavailablePQCProvider`. It throws on all operations if no PQC implementation is available.

Terminology alignment. We use two orthogonal knobs: provider selection (`preferPQC`/`requirePQC`) and handshake fallback policy (`default`/`strictPQC`). In our evaluation, `default` corresponds to prefer-PQC with evented classic fallback permitted, while `strictPQC` corresponds to requiring PQC and forbidding fallback.

3.7 KEM-DEM Authenticated Encryption Format

We use a KEM-DEM envelope (KEM $\rightarrow$ HKDF $\rightarrow$ AEAD) as a compatibility layer when native HPKE is unavailable, binding role and transcript into the key schedule with explicit length limits for DoS protection. On Apple 26+ platforms, we use native CryptoKit PQC primitives (ML-KEM/ML-DSA) within the same envelope to preserve a single wire format across platforms. Full header/body formats, versioning, and length limits are provided in the Supplementary Materials (KEM-DEM Envelope).

4 HANDSHAKE STATE MACHINE
4.1 Design Principles

The handshake subsystem addresses three critical requirements:

1) **Race Condition Prevention:** Concurrent message arrival and timeout expiration must not cause double-resume of continuations.
2) **Sensitive Material Protection:** Ephemeral private keys and shared secrets must be zeroized on all exit paths.
3) **Observability:** All state transitions and failures must emit structured events.

4.2 Actor-Isolated Architecture

The `HandshakeDriver` actor manages handshake state with compile-time data race prevention:

The `HandshakeContext` actor isolates sensitive cryptographic material:

Reentrancy Considerations. Swift actors permit method re-entry at suspension points (`await`). While actor isolation prevents

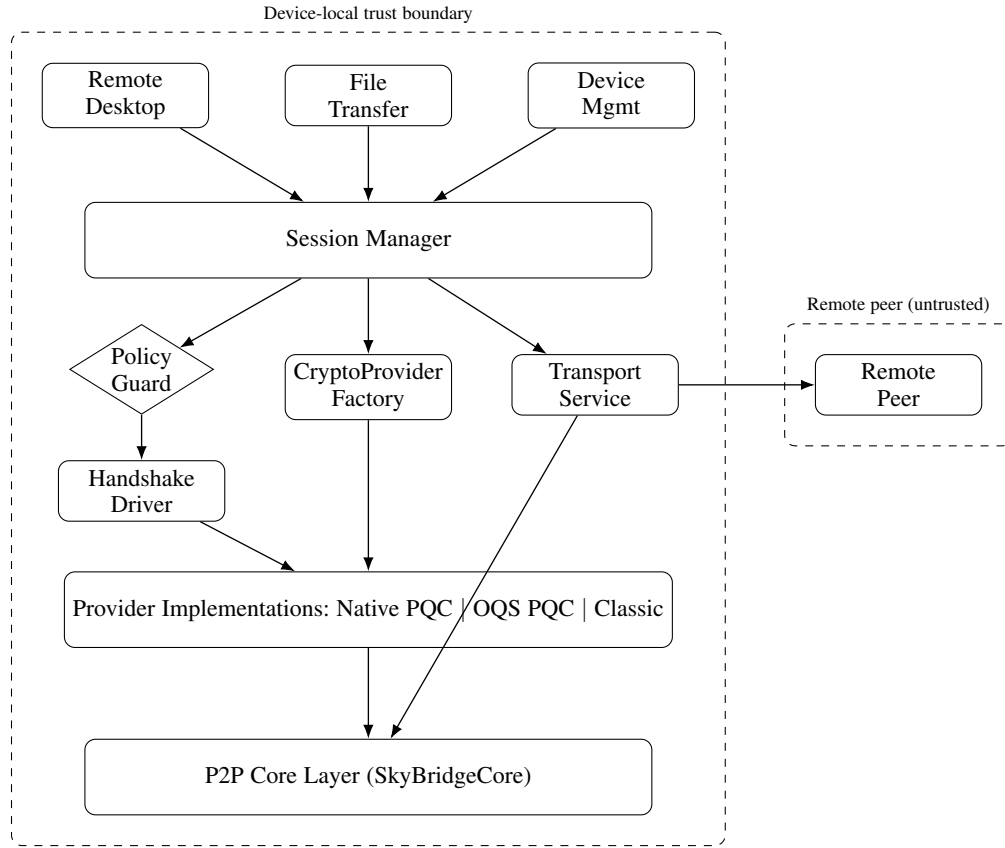| Initiator attempt(s)	Responder Capability	Outcome	Notes
Attempt 1: PQC-only	PQC available	PQC established	Best available selected
Attempt 1: PQC-only; Attempt 2: classic-only	Classic only (no PQC material)	Classic established	Graceful fallback (evented)
PQC-only (`strictPQC`)	Classic only	Handshake failed	Policy enforced (no fallback)
Classic-only	PQC available	Classic established	Initiator policy respected

Fig. 2. SkyBridge Compass system architecture showing device-local trust boundary (dashed) and policy guard (diamond).

data races, it does not prevent interleaving of method executions. Our `HandshakeDriver` mitigates reentrancy risks by routing completion through `finishOnce(with:)`, buffering early results, canceling timeouts on completion, and advancing state before `await` points.

Await-window limitation. The `handleMessageB` method awaits `cryptoProvider` operations inside the actor. If a new message arrives during this await, the actor could theoretically process it before the current operation completes. We mitigate this by advancing state to `.processingMessageB` before the await, copying immutable inputs, validating a monotonic epoch counter after resumption, and emitting security events on unexpected transitions.

This approach provides structural protection against reentrancy hazards rather than relying on timing assumptions. A formal analysis using model checking is planned for future work.

4.3 State Transitions

Fig. 3 illustrates the handshake state machine for both roles.

4.4 Double-Resume Prevention

The `finishOnce` method provides a single convergence point for handshake completion, handling the race between timeout expiration and message arrival by: (1) canceling the timeout task on any completion, (2) guarding continuation access with immediate nil assignment, and (3) buffering early results for late continuation establishment.

4.5 Secure Zeroization

Swift's standard `Data` and `Array` types use copy-on-write (COW) semantics, which can leave uncontrolled copies of sensitive material in memory. The `SecureBytes` class addresses this by using manual `UnsafeMutableRawPointer`
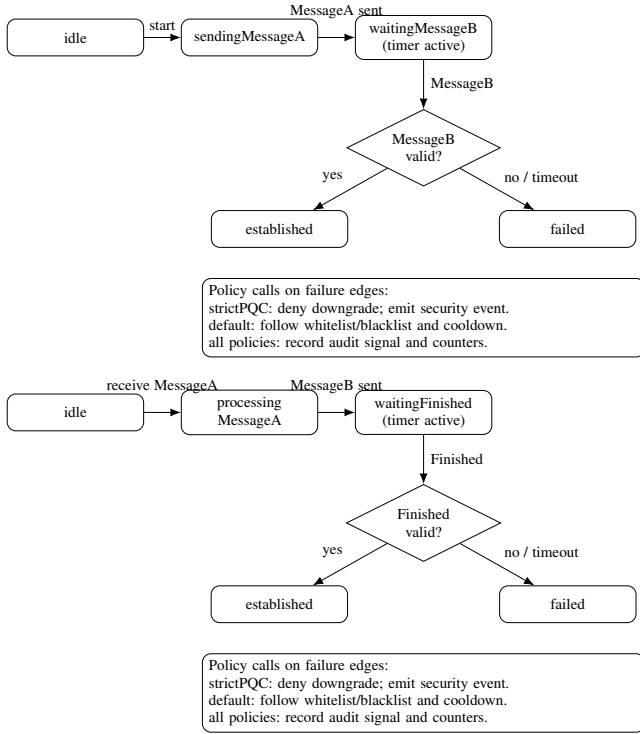
Fig. 3. Handshake state machines for Initiator (top) and Responder (bottom).

allocation to maintain exclusive ownership, preventing COW-induced secret proliferation. All failure paths explicitly call `context.zeroize()` before error propagation, with `deinit` zeroization as defense-in-depth. On Darwin platforms, zeroization uses platform-supported secure wipe primitives (e.g., `explicit_bzero` or `memset_s`) where available. We avoid exporting secrets as `Data` when possible, but acknowledge that OS-level snapshots, API boundary copies, or privileged inspection can still retain residues; zeroization is a defense-in-depth measure rather than a sole security boundary.

4.6 Transcript Binding and Signature Separation

The protocol uses domain-separated signatures to prevent cross-message and cross-session replay attacks.

Signature A (MessageA) is computed by the Initiator over the MessageA-local fields, binding suite preference, policy intent, and identity context before any responder data is received:

Signature A preimage includes the domain separator `SkyBridge-A`, the version, canonical supported suites and key shares, the client nonce, capabilities, policy, and the initiator identity public key; optional Secure Enclave binding uses `SkyBridge-SE-A`.

Signature B (MessageB): The Responder signs a transcript that commits to MessageA:

Signature B preimage includes the domain separator `SkyBridge-B` plus `transcriptA`, selected suite wire ID, responder key share, server nonce, payload hash, and the responder identity public key; optional Secure Enclave binding uses `SkyBridge-SE-B`.

Key Derivation: Nonces, transcripts, suite binding, and *directional separation* are included in the KDF, and the salt is transcript-bound (non-empty):

KDF input concatenates the `SkyBridge-KDF` label, suite wire ID, transcript hashes, and nonces. The salt uses the domain string `SkyBridge-KDF-Salt-v1|`. Direction labels separate traffic classes:

- `initiator_to_responder`
- `responder_to_initiator`

This structure is designed to ensure: (1) `sigB` commits to `MessageA`, so any tampering breaks verification; (2) domain separators prevent cross-protocol attacks; (3) SE signatures are session-bound and non-replayable.

5 SECURITY MODEL AND GUARANTEES

5.1 Security Model

We assume an active network adversary with full control of the discovery channel (drop, delay, reorder, duplicate, modify, and inject messages) but without breaking standard cryptographic assumptions. The attacker can replay prior handshakes and attempt downgrade by suppressing PQC-related fields. We assume the out-of-band pairing ceremony is trusted and that device key storage (Keychain/Secure Enclave) preserves key integrity. The attacker may compromise a peer after pairing and learn long-term signing key material, but cannot retroactively forge signatures for past sessions. We do not claim forward secrecy for KEM-based suites in v1: ML-KEM encapsulation targets a peer's long-term KEM public key obtained during pairing. Accordingly, post-quantum confidentiality against store-now-decrypt-later adversaries holds only while long-term KEM private keys remain uncompromised; later compromise of the corresponding KEM private key can enable decryption of recorded sessions.

Assumptions:

- The pairing ceremony delivers the correct peer identity key (PIN or visual verification is not bypassed).
- Pinned identity keys stored in Keychain/Secure Enclave retain integrity across normal device operation.
- Devices remain uncompromised (no root or malware control) during handshake execution.

If assumptions fail:

- If pairing is subverted and the pinned identity key is replaced, peer authentication is no longer meaningful; the system must require re-pairing and treat the session as untrusted.
- If device integrity is lost, confidentiality and key secrecy cannot be guaranteed; the system should abort and emit high-severity events such as `identityMismatch` or `handshakeFailed`.

5.2 Security Goals and Proof Sketches

Definitions.

- **D1 (Forced downgrade / forced fallback).** A downgrade occurs if a session is established with a classic suite when policy requires PQC (`strictPQC`), or if classic fallback is accepted under the default policy without satisfying the local-only fallback gate (Definition D2) and without emitting the corresponding audit event (Property G8).
- **D2 (Local-only fallback gate).** Fallback is permitted only after (i) the peer identity is verified against the

pinned key (paired peers) and (ii) the triggering failure is locally generated capability/instantiation unavailability (whitelist), excluding timeout and any unauthenticated network-derived failure.

- **D3 (Audit evidence).** The minimal sufficient evidence set is the tuple {event type, timestamp, deviceId, suiteId, failure reason code, policy mode, fallback flag, cooldown remaining} (Property G8).

We state the core properties as invariants and provide proof sketches tied to protocol mechanics. To strengthen reviewer-facing claims, we additionally provide a minimal symbolic model artifact (Tamarin) that captures suite negotiation, transcript binding, explicit key confirmation, and the fallback gate; lemmas are machine-checkable from the released artifact (`formal/skybridge_minimal.spthy`). Full formalization of the entire Swift implementation (including concurrency and platform APIs) remains future work.

Property G1 (Negotiation Integrity): The selected suite must be among the Initiator's offered suites and have a corresponding key share.
Proof sketch:

- `MessageB` includes `selectedSuite`, and `sigB` covers `MessageA`.
- Any modification to the offered suite list or key-share list breaks `sigB` verification.
- The Responder validates `selectedSuite` against the offered suites and checks that a matching key share is present.

Property G2 (Mutual Authentication for Paired Peers): For previously paired devices, the peer identity is authenticated and bound to the handshake transcript.
Proof sketch:

- `identityPubKey` is pinned during OOB pairing.
- `MessageA`/`MessageB` signatures over transcript data must verify under the pinned key; otherwise `identityMismatch` aborts the handshake.

Property G3 (Session Key Secrecy): The derived session keys remain secret under standard KEM/DH and signature security assumptions. For KEM-based PQC suites in v1, we do not claim post-compromise forward secrecy: confidentiality holds only while long-term KEM private keys remain uncompromised.
Proof sketch:

- The shared secret is derived from KEM decapsulation or DH, and keys are extracted via HKDF with transcript-bound salt.
- An attacker without the private key material cannot compute the shared secret or derive session keys.

Property G4 (Replay Resistance): Replayed handshakes do not establish a session.
Proof sketch: `handshakeId` derives from fresh nonces and suite identifiers; replay detection caches recent IDs and rejects duplicates within the window.

Property G5 (Downgrade Resistance under strictPQC): No fallback to classic is allowed under strict policy.
Proof sketch: `TwoAttemptHandshakeManager` enforces a policy gate before any fallback attempt; strictPQC forbids all fallback edges regardless of error type. This is validated by policy downgrade benchmarks (Fig. 4).

Property G6 (Safe Fallback under default policy): Fallback can only occur after peer identity is verified under the pinned key and only for a whitelisted set of locally generated capability errors; timeout and network-derived failures never trigger fallback.
Proof sketch: Fallback is gated after identity verification; the whitelist is evaluated on locally generated capability/instantiation errors rather than unauthenticated network input. `timeout` is explicitly blocked and per-peer cooldown limits repeated downgrades. *Proposition (No forced fallback by packet loss):* Under the attacker model in Section 5, an active network attacker who can drop/delay/reorder messages cannot induce a successful classic session by causing timeouts, because timeout-triggered fallback is disallowed and the default-policy fallback gate (Definition D2) requires locally generated unavailability after identity verification.

Property G7 (Legacy Acceptance Preconditions): Legacy P-256 signatures are accepted only under authenticated pairing or an existing trust record.
Proof sketch: Legacy acceptance is gated by `LegacyTrust Precondition`; pure network stranger connections fail. This is validated by the precondition test matrix (Table 6).

Property G8 (Auditability): Cryptographic downgrades and exceptional states are observable via a minimal sufficient information set.
Proof sketch: The handshake emits `cryptoDowngrade` events with reason, deviceId, and cooldown context, enabling post-hoc verification of policy adherence.
Audit field requirements:

- **MUST log** (minimal sufficient set): event type, timestamp, deviceId, suiteId, failure reason code, policy mode, fallback triggered (bool), cooldown remaining.
- **MUST NOT log** (privacy preservation): session keys, nonces, raw signatures, message payloads, user identifiers beyond deviceId.
- **Tamper-evidence:** Events are emitted synchronously before state transitions complete; the resulting trace is checkable against expected state-machine templates (used in our audit-signal fidelity tests). Log collection relies on OS unified logging (`os_log`) for best-effort retention and is not a cryptographic tamper-proof store; privileged adversaries can alter or truncate logs.

6 IMPLEMENTATION

Implementation references (source files, line numbers) are consolidated in the Supplementary Materials (Artifact Map).

6.1 Platform Support Matrix

SkyBridge Compass uses a tiered provider model: a native PQC provider when available (e.g., CryptoKit ML-KEM/ML-DSA), a portable liboqs provider, and a classic provider. Provider selection is deterministic and prefers the strongest available tier; Secure Enclave proof-of- possession is optional and orthogonal to the negotiated suite. The full platform support matrix and suite availability are in the Supplementary Materials (Platform Support Matrix).

6.2 Native PQC Integration

The native provider wraps CryptoKit ML-KEM/ML-DSA APIs [23] when the SDK exposes them, enabling platform-backed PQC

TABLE 3

Property	Enforced at	Evidence
G1 Negotiation integrity	Suite/keyshare validation + transcript-bound `sigB`	Fig. 1, HandshakeDriverTests
G2 Mutual authentication	Identity pinning + signature verification	Table 5 (wrong signature), Table 6
G3 Session key secrecy	KEM/DH + HKDF with transcript-bound salt	Property 1, Supplementary Materials (ML-DSA Key Sizes)
G4 Replay resistance	`handshakeId` cache + nonce binding	Property-Oriented Testing (Section 7)
G5 strictPQC no-downgrade	Policy gate in TwoAttemptHandshakeManager	Fig. 4, PolicyDowngradeBenchTests
G6 Default safe fallback	Whitelist/blacklist + local capability gate + cooldown	Fig. 5, Table 5
G7 Legacy acceptance precondition	LegacyTrustPrecondition	Table 6
G8 Auditability	Security event emission	Fig. 6, Table 7

suites and HPKE-based encapsulation. We treat this as a deployment instance rather than a protocol dependency; integration details are provided in the Supplementary Materials (Native PQC Integration).

6.3 Conditional Compilation Strategy

PQC symbols are gated behind build-time flags to keep the codebase buildable on SDKs that lack PQC types and to avoid compile-time failures. The flagging strategy and build settings are detailed in the Supplementary Materials (Conditional Compilation).

6.4 Security Event Emission

All cryptographic decisions emit structured events for auditability. The emitter provides backpressure and rate limiting; implementation details are in the Supplementary Materials (Security Event Emission).

6.5 Secure Enclave Integration

For hardware-backed signing, a `SigningCallback` protocol allows Secure Enclave keys without exposing private material. Availability, fallback behavior, and key-type details are provided in the Supplementary Materials (Secure Enclave Integration).

6.6 Signing Key Hierarchy

The system uses two complementary signatures: (1) protocol signatures via the negotiated suite (Ed25519 for classic, ML-DSA-65 for PQC) to bind peer identity to the transcript, and (2) optional Secure Enclave P-256 signatures for device proof-of-possession. Protocol signatures are mandatory; Secure Enclave PoP is optional and provides a hardware-backed signal. Key hierarchy details and verification rules are provided in the Supplementary Materials (Signing Key Hierarchy).

6.7 Pre-Negotiation Signature Algorithm Selection and Two-Attempt Strategy

A fundamental challenge is that `sigA` must be generated before negotiation completes, yet the signature algorithm should match the negotiated suite. We resolve this with pre-negotiation signature selection and a two-attempt strategy. Each attempt is homogeneous (PQC-only or classic-only), and the signature algorithm is chosen based on the offered suites. The initiator first attempts PQC, then falls back to classic only for a whitelist of benign, locally generated capability errors after peer identity verification; timeout, signature verification, identity mismatch, replay, and other network-derived failures never trigger fallback. This policy is

enforced by the handshake manager and logged as security events. Detailed error lists and type-level enforcement are provided in the Supplementary Materials (Two-Attempt Strategy).

Fallback Boundary Conditions (Explicit Scope):

- *We defend against:* (1) Attacker-induced timeout forcing downgrade (blocked by timeout blacklist); (2) Rapid downgrade cycling via repeated failures (5-minute per-peer cooldown); (3) Signature/identity attacks masquerading as capability failures (verification failures are blacklisted).

- *We do not defend against:* (1) Attacker persistently blocking PQC traffic for >5 minutes (availability vs. security trade-off; strictPQC mode provides full protection at cost of connectivity); (2) Legitimate PQC provider failures causing temporary classic fallback (by design, to maintain usability).

- *Cooldown rationale:* We set a conservative default of 5 minutes to bound downgrade cycling while limiting availability impact; the cooldown is a tunable policy knob, and strictPQC users requiring zero-downgrade guarantees should disable fallback entirely.

Attempt State and Identifiability:

- *Shared across attempts:* pinned identity fingerprint and pinned KEM public keys (trust record), and per-peer fallback cooldown state.

- *Not shared across attempts:* nonces, transcript hashes, handshakeId/replay tags, ephemeral key shares, and session keys; each attempt runs a fresh state machine.

- *Identifiability:* a passive observer can distinguish a classic fallback attempt from a PQC attempt by wire format. We do not transmit an explicit attempt counter, and strictPQC disables fallback when capability privacy outweighs compatibility.

Security Events:

Every fallback emits a `cryptoDowngrade` event with full context (reason, deviceId, cooldown), enabling audit and anomaly detection.

7 EVALUATION

7.1 Experimental Setup

We evaluate SkyBridge Compass along two primary tracks: **security-centric evidence** (fault injection, downgrade suppression, legacy precondition enforcement, and auditability) and **cost-centric evidence** (handshake latency, wire overhead, provider selection overhead, and data-plane throughput).

Environment: All experiments are run on Apple Silicon (M1/M3 class) with macOS 26.x, where CryptoKit PQC is available when the SDK exposes ML-KEM/ML-DSA types. Build configuration uses Release (`-O`) with non-essential logging disabled. Timing uses a monotonic clock (`ContinuousClock`) with N=1000 iterations after 10 warmup runs; reported percentiles (p50/p95/p99) are computed directly from samples.

Repeatability: We support multi-batch repeatability by restarting the benchmark test process for each batch. Supplementary Tables S8–S9 report the number of observed batches (*B*) and (when $B \geq 2$) mean and 95% confidence intervals across batches. To reproduce multi-batch CIs, re-run the harness with `SKYBRIDGE_BENCH_BATCHES=3–5`.

Reproducibility: We ship an opt-in benchmark suite under `Tests/SkyBridgeCoreTests/`. For a one-shot run, use `Scripts/run_paper_eval.sh`. The complete command reference and CSV artifact specifications are provided in the Supplementary Materials (Reproducibility).

iOS Reproducibility: Because CI-style automation for iOS PQC measurements is sensitive to device availability and simulator runtime support, we also ship an on-device micro-benchmark in the iOS app that exports JSON artifacts (timestamped) for Supplementary tables. This enables independent reproduction of KEM, signature, and KEM–DEM costs on commodity iOS 26+ hardware without modifying the protocol code paths.

Interpretation: The loopback microbench isolates cryptographic and state-machine overhead; it does not model OS network stacks, NAT traversal, or heterogeneous access networks. We report simulated impairments separately (Section 7.5) and list real-network end-to-end measurement as future work (Section 8).

7.2 Security Validation

We validate security properties through fault injection, property-based testing, and audit-signal fidelity analysis. This section presents the key results; detailed test matrices and methodology are provided in the Supplementary Materials (Regression Testing).

Failure-Mode Robustness. The actor-isolated handshake driver targets deterministic failure semantics; in our fault-injection tests we observed no unhandled errors, double-resume, or sensitive-material leaks. We inject faults including timeout, malformed framing, invalid signatures, out-of-order delivery, and concurrent cancellation (n=1000 per scenario per policy).

Unknown-suite handling: We parse unknown suite identifiers as an explicit `unknown(wireId)` value and reject them if they appear in `keyShares` or as `selectedSuite`. This prevents silent "unknown → classic" coercions that would otherwise weaken downgrade guarantees.

The delay-within-timeout and delay-exceed-timeout scenarios approximate RTT jitter and loss-induced stalls, while drop and duplicate cases approximate packet loss and reordering. The per-scenario event distributions are recorded in `Artifacts/fault_injection_2026-01-16.csv`.

Downgrade events are quantified separately in the policy bench (Fig. 4) to avoid conflating fallback behavior with transport corruption scenarios.

The plot in Fig. 4 uses the policy-downgrade dataset. File: `policy_downgrade_2026-01-16.csv`. It demonstrates that strictPQC never emits fallback events even under forced PQC-unavailable errors; default policy shows non-zero downgrade events.
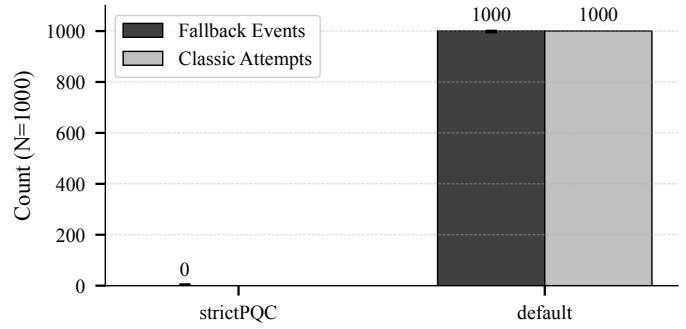

Fig. 4. Policy guard enforces strictPQC: fallback events per 1000 runs, macOS 26.x, N=1000 per policy. Error bars show 95% Wilson binomial CI.
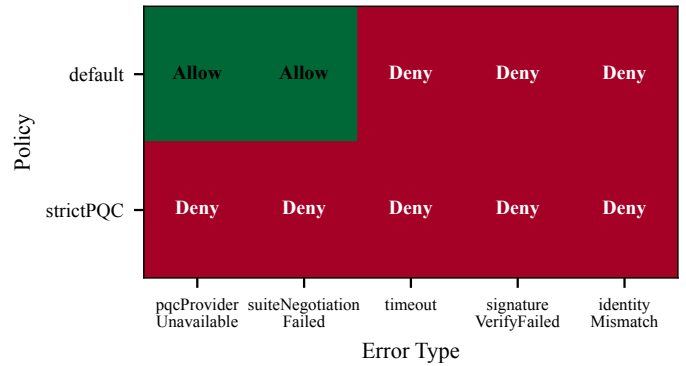

Fig. 5. Downgrade decision matrix (policy × error) with explicit allow/deny semantics, macOS 26.x. Each cell is labeled ALLOW or DENY for grayscale readability.

Fig. 5 encodes the explicit downgrade whitelist/blacklist: only PQC-unavailability and suite-selection errors may fallback under default policy; strictPQC denies all fallback edges.

Fig. 6 combines fault-injection counts with the downgrade-acceptance event from the policy bench to visualize observability of failures and downgrades across policy modes.

Property-Oriented Testing. Correctness properties are validated with property-based tests using reproducible seeds (0xDEADBEEF). Key properties include: (1) KEM-DEM round-trip correctness across all providers (100 trials), (2) signature verification integrity (100 payloads per provider), (3) secure zeroization on deallocation, and (4) state-machine single-resume
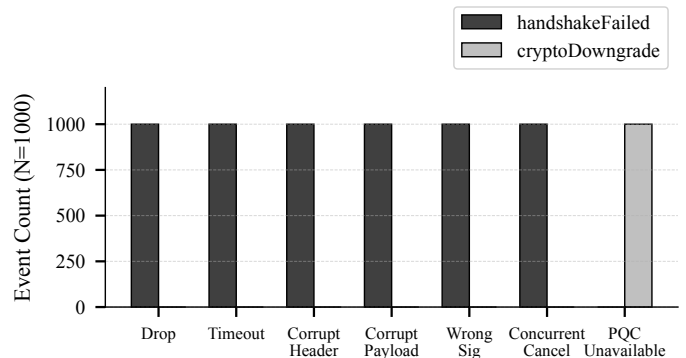

Fig. 6. Failure-mode histogram of security events, macOS 26.x, N=1000 per scenario.

Configuration	KEM encaps. (mean/p99)	Sign (mean/p99)	Verify (mean/p99)	KEM–DEM seal+open (mean/p99)
Classic (X25519 + Ed25519)	0.414 / 0.704	0.122 / 0.126	0.085 / 0.133	0.389 / 0.534
CryptoKit PQC (ML-KEM-768 + ML-DSA-65)	0.149 / 0.190	1.072 / 2.108	0.048 / 0.055	0.245 / 0.272
CryptoKit Hybrid (X-Wing + ML-DSA-65)	0.282 / 0.452	1.403 / 2.654	0.048 / 0.053	0.324 / 0.391

Failure Mode	Runs	No Unexp. Error	No Double Resume	Zero Verified	$E_{\texttt{handshakeFailed}}$	$E_{\texttt{cryptoDowngrade}}$
Out-of-order	1000	1	1	1	0	0
Duplicate	1000	1	1	1	0	0
Drop	1000	1	1	1	1000	0
Delay within timeout	1000	1	1	1	0	0
Delay exceed timeout	1000	1	1	1	1000	0
Corrupt header	1000	1	1	1	1000	0
Corrupt payload	1000	1	1	1	1000	0
Wrong signature	1000	1	1	1	1000	0
Concurrent cancel	1000	1	1	1	1000	0
Concurrent timeout	1000	1	1	1	1000	0

guarantee under concurrent stress (500 iterations). A standalone Python decoder (`Scripts/wire_format_sanity.py`) provides non-Swift interoperability validation of wire formats.

ML-DSA Key Lifecycle. ML-DSA-65 key generation, storage, and sign-verify operations are validated against FIPS 204 specifications [24]. Size compliance results are reported in the Supplementary Materials (ML-DSA Key Sizes); all property tests (round-trip, tampering detection, key independence) achieve 100% pass rate. Keys are stored in Keychain with `kSecAttrAccessibleAfterFirstUnlock ThisDeviceOnly` and `kSecAttrSynchronizable= false` to ensure device-local protection.

Legacy Fallback Security. Legacy P-256 signatures are only accepted when a security precondition is satisfied: (1) authenticated out-of-band channel (QR code, PAKE/PIN), or (2) existing TrustRecord with `legacyP256PublicKey`. Pure network stranger connections are rejected to prevent downgrade attacks.

Property 7 (Legacy Fallback Security Precondition): A legacy P-256 signature SHALL only be accepted when a security precondition is satisfied. Pure network stranger connections SHALL be rejected. Validated with 12 test cases covering all precondition combinations (100% coverage of the precondition state space).

Audit-Signal Fidelity. Security events provide deterministic, audit-friendly (best-effort, non-cryptographic) telemetry of failures and policy decisions. Fig. 7 illustrates a representative timeout trace showing attacker action, policy gate response, and emitted event.

Evaluation approach: We evaluate audit-signal fidelity using a spec-to-event mapping consistency metric, *not* a statistical classifier. Each fault-injection scenario deterministically maps to exactly one expected trace template (event sequence), e.g., timeout → `handshakeFailed=1`, `cryptoDowngrade=0`. Match rate (MR)=1.00 means all N iterations matched the expected template; spurious rate (SR)=0.00 means no unexpected events were emitted. This is regression consistency against the protocol specification, not ML classification performance.

Table 7 summarizes results across all fault-injection scenarios: all scenario classes achieve 1.00 match and 0.00 spurious rates, confirming correct event semantics.

Due to space constraints, the complete regression test matrix, TLV encoding validation, and implementation component details are provided in the Supplementary Materials (Regression Testing and Artifact Map).

7.3 Performance Benchmarks

Table 8 consolidates all performance metrics. We measure end-to-end handshake completion time using an in-memory loopback transport to isolate cryptographic and state-machine

TABLE 6
Legacy Fallback Precondition Test Results. All tests executed via LegacyFallbackPreconditionTests. Tests validate Property 7: Legacy Fallback
Security Precondition.

Scenario	Precondition	Expected Result	Actual Result
Pure network stranger	None	Reject	OK Reject
QR code pairing (verified)	authenticatedChannel	Allow	OK Allow
PAKE pairing (verified)	authenticatedChannel	Allow	OK Allow
Existing TrustRecord with legacy key	existingTrustRecord	Allow	OK Allow
Existing TrustRecord without legacy key	None	Reject	OK Reject
Network discovery	None	Reject	OK Reject
Unverified authenticated channel	None	Reject	OK Reject

TABLE 7
Audit-Signal Fidelity Summary. Match/spurious rates reflect *deterministic* protocol event mapping (each scenario maps to exactly one expected
trace template), not statistical classification. Perfect scores indicate regression consistency against the protocol specification.

Scenario Class	Expected Signal	N	Match (MR)	Spurious (SR)
Drop/Timeout (default)	handshakeFailed=1, cryptoDowngrade=0	3000	1.00	0.00
Drop/Timeout (strictPQC)	handshakeFailed=1, cryptoDowngrade=0	3000	1.00	0.00
Corrupt/Wrong Sig	handshakeFailed=1, cryptoDowngrade=0	6000	1.00	0.00
Ordering Benign	handshakeFailed=0, cryptoDowngrade=0	6000	1.00	0.00
PQC unavailable (default)	cryptoDowngrade=1	1000	1.00	0.00
PQC unavailable (strictPQC)	cryptoDowngrade=0	1000	1.00	0.00

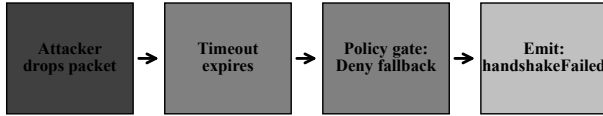
Fig. 7. Audit-signal event trace (representative timeout case) showing attacker action, policy gate, and emitted event.

Fig. 8. Handshake latency percentiles for Classic vs liboqs PQC vs CryptoKit PQC (N=1000). Generated from `Artifacts/handshake_bench_2026-01-16.csv`; values match the Performance Summary (Table 8) and Supplementary Table S1.

overhead. Each configuration reports distributional statistics over N=1000 iterations after 10 warmup runs.

Latency Distribution. Fig. 8 shows latency percentiles across configurations. Classic handshakes remain under 2 ms at p99; CryptoKit PQC stays under 6.4 ms at p99 with variance dominated by ML-DSA signature verification overhead. Extended statistics (mean/p50/p95/p99, std) are available in Supplementary Tables S1–S2 (submitted as separate file `supplementary.pdf`).

Wire-Format Breakdown. The $15\times$ size increase from Classic (827 B) to PQC (12.2 kB) is dominated by ML-DSA signatures (3309 B) and ML-KEM ciphertexts (1088 B). Fig. 9 shows the per-field breakdown.

Note: X-Wing hybrid KEM adds 32 B to MessageA's keyshare (1120 B vs 1088 B for ML-KEM-768), yielding a measured total handshake size of 12,195 B; detailed field breakdowns are available in Supplementary Table S3 (`MessageA.XWing/MessageB.XWing`).

Data-Plane and Provider Selection. Post-handshake symmetric AEAD (AES-256-GCM) throughput reaches 3.7 GB/s for 1 MiB payloads on Apple Silicon, with CPU cost of 0.25 ns/byte. In v1 the data-plane AEAD is fixed to AES-256-GCM and
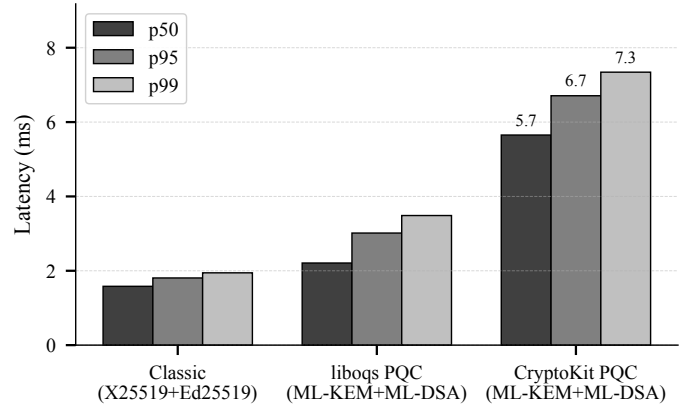
recorded separately from the handshake suite; thus throughput is independent of the negotiated KEM/signature suite. Provider selection overhead is sub-microsecond for cached paths (0.17 μs p50) and under 6 μs even for fallback scenarios. Detailed breakdowns are provided in the artifact CSV files.

7.4 System-level Impact and Amortization

The preceding microbenchmarks isolate handshake and cryptographic costs. To align evaluation with the *remote desktop / file transfer* framing, we additionally measure user-visible, session-level metrics and quantify how one-time handshake cost amortizes over longer sessions and larger transfers.

Metrics: (i) `T_connect` (connect start $\rightarrow$ handshake established, including explicit Finished key confirmation and event emission), (ii) `T_first_frame` (connect start $\rightarrow$ first decrypted

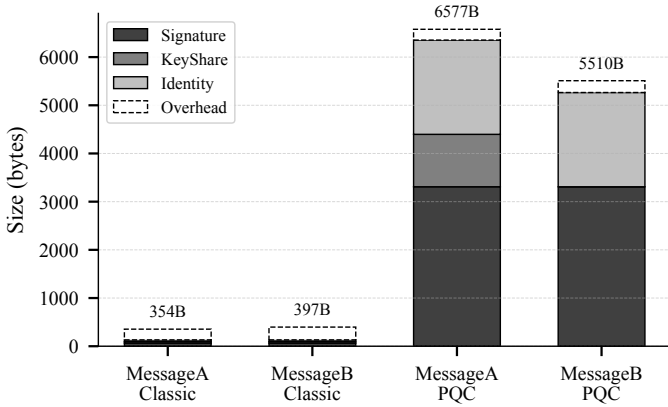| Configuration	Handshake Latency		RTT		Wire Size	Throughput
	mean (ms)	p95 (ms)	p50 (ms)	p95 (ms)	(bytes)	(GB/s)
Classic	1.62	1.81	0.41	0.46	827	3.7
liboqs PQC	2.29	3.01	0.80	1.35	12,163	3.7
CryptoKit PQC	5.76	6.71	1.59	2.27	12,163	3.7

Fig. 9. Handshake message size breakdown (signature, keyshare, identity fields, framing overhead) for MessageA/MessageB across Classic, PQC (liboqs), and PQC (CryptoKit) configurations. Counts reflect MessageA/B payloads only; the *payload-only* handshake size adds two 38 B Finished frames (accounted for in Table 8), and detailed field sizes are reported in Supplementary Table S3. Generated from `Artifacts/message_sizes_2026-01-16.csv`.

Fig. 10. System-level impact and amortization (p95). Top: `T_connect` across RTT+jitter conditions. Bottom: total session time (`T_total` = connect + transfer) versus file size under RTT 50±20 ms, showing that one-time handshake overhead is amortized by larger transfers. Data from `Artifacts/system_impact_2026-01-22.csv`.

remote-desktop frame processed by the rendering path), and (iii) `T_total` (connect start → completion of a bulk transfer over the negotiated session keys).

Method: We reuse the production `HandshakeDriver` and negotiated `SessionKeys` and execute (a) one encrypted "first frame" payload and (b) bulk encrypted transfer workloads of 1–100 MiB. Network conditions are emulated as deterministic RTT+jitter on a reliable stream; large data-plane payloads are bandwidth-limited (MiB/s) to avoid per-chunk RTT artifacts while remaining reproducible. Results are emitted to `Artifacts/system_impact_2026-01-22.csv` by `SystemImpactBenchTests`.

Traffic Analysis Mitigation (SBP2 Padding). To reduce attempt identifiability on the wire, SkyBridge optionally applies a constant-size framing layer (SBP2) that buckets framed payloads into powers-of-two-sized envelopes. Here, **SBP1** denotes *handshake-frame padding* (MessageA/MessageB/Finished are padded to fixed buckets before transport), while **SBP2** denotes *traffic framing and padding* applied to all framed payloads (control and encrypted data-plane) with a per-frame header and bucketed padding up to a configured cap. We measure SBP2 behavior on a real protocol trace: an end-to-end handshake (SBP1 + SBP2 enabled) on an in-memory loopback transport, followed by bidirectional AES-GCM–encrypted data-plane frames derived from the negotiated session keys. Supplementary Table S4 reports the resulting quantization behavior and overhead; the full per-label bucket histograms are included in `Artifacts/traffic_`
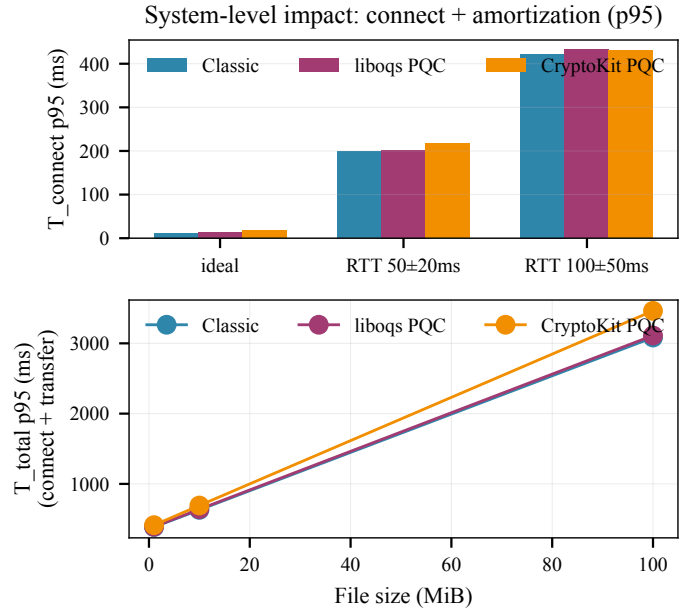
`padding_2026-01-16.csv`.

Sensitivity Study (Bucket Cap). We vary the maximum bucket size for SBP2 (*cap* = 64 KiB, 128 KiB, 256 KiB) and re-run the same deterministic protocol trace. We report (i) padding overhead and (ii) *over-cap rate* (fraction of frames whose framed payload exceeds the cap and thus cannot be further bucketed), and use *bucket entropy* (Shannon entropy of padded bucket sizes per label) as a privacy proxy for size quantization strength. Supplementary Table S5 summarizes the overhead–privacy tradeoff and distinguishes remote-desktop-like vs file-chunk-like large-frame distributions.

7.5 External Validity Under Network Impairments

We simulate four network conditions with a retry-aware model (max 2 retries) to approximate loss, jitter, and reordering. The simulator is in-process (not kernel `netem`): loss is independent Bernoulli per packet; jitter is sampled uniformly from ±J ms around base RTT; and reordering is modeled as an added random delay (50–150 ms) applied to 10% of packets. Table 11 summarizes completion and p95 latency ranges across classic/liboqs/CryptoKit suites using `Artifacts/network_condition_`

TABLE 9
System-level impact and amortization (p50/p95, ms). $T_{connect}$ is measured from connect start to handshake established (including Finished key confirmation and event emission). T_{total} is measured from connect start to completion of an encrypted bulk transfer. Network conditions emulate deterministic RTT+jitter on a reliable stream; large data-plane payloads are bandwidth-limited (MiB/s) to avoid per-chunk RTT artifacts. Runs: $N_{connect}$=1000; N_{total} (RTT 50±20 ms) = 1MiB:N=1000, 10MiB:N=500, 100MiB:N=100. Data from `Artifacts/system_impact_<date>.csv`.

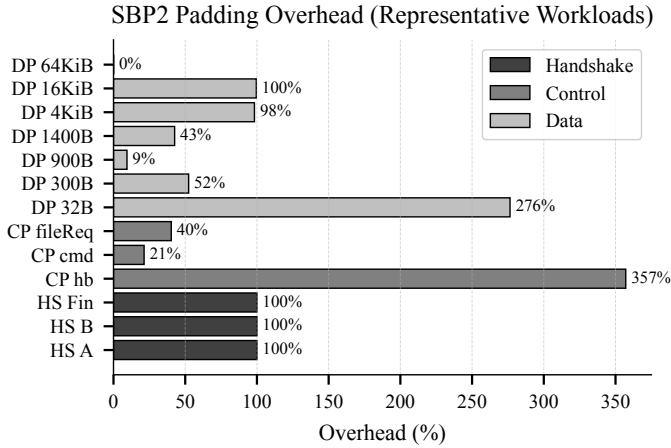|	$T_{connect}$ (p50/p95)			T_{total} (p50/p95), RTT 50±20 ms		
Suite	ideal	RTT 50±20 ms	RTT 100±50 ms	1 MiB	10 MiB	100 MiB
Classic	11.4/12.0	187.7/193.5	411.5/422.4	372.6/386.3	613.8/630.6	3038/3083
liboqs PQC	13.1/14.3	190.9/196.2	421.8/432.8	379.2/390.6	619.7/639.5	3055/3111
CryptoKit PQC	16.0/18.2	197.1/215.2	417.6/430.8	382.0/412.9	626.8/692.4	3062/3462

Fig. 11. SBP2 padding overhead across representative handshake, control, and data-plane workloads (macOS 26.x). Overhead is computed as `(paddedBytes/rawBytes − 1) × 100%` aggregated per label over a deterministic trace (N=1000 handshake iterations; data-plane workload uses encrypted control messages and large-frame mixes). Generated from `Artifacts/traffic_padding_2026-01-16.csv`.
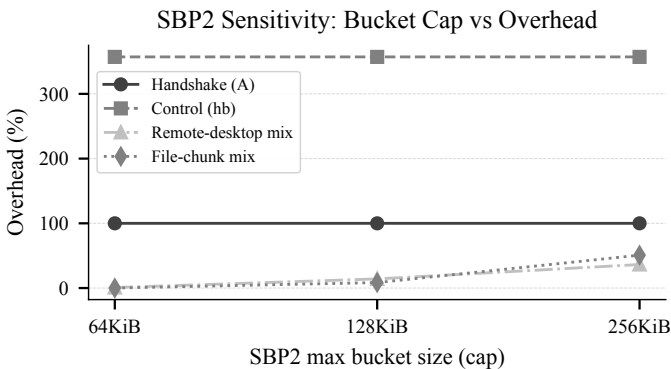

Fig. 12. SBP2 bucket-cap sensitivity (macOS 26.x). Curves show padding overhead for representative large-frame distributions (remote-desktop mix vs file-chunk mix) as the cap increases from 64 KiB to 256 KiB; lower over-cap rate implies broader padding coverage. Generated from `Artifacts/traffic_padding_sensitivity_2026-01-16.csv`.

TABLE 10
Handshake vs data-plane AEAD selection (v1).

Layer	AEAD	Negotiation source
Handshake payloads	AES-256-GCM	Suite tuple
Data-plane traffic	AES-256-GCM	Fixed (v1)

`2026-01-11.csv` generated by `Scripts/run_network_bench_retry.swift`. Even under severe loss, completion remains above 99.8%, and `handshakeFailed` events track the few failures; no downgrade events occur because fallback is only permitted for PQC-unavailability (Fig. 4).

8 LIMITATIONS AND FUTURE WORK

8.1 Current Limitations

1) **iOS PQC Availability (iOS < 26):** We do not currently bundle liboqs for iOS; therefore PQC suites are only available on iOS 26+ devices via CryptoKit PQC. To support reproducible mobile evaluation, we ship an *on-device* self-test and micro-benchmark harness in the iOS app (Settings → PQC → "Self-test/Bench"), which exports JSON artifacts for inclusion in Supplementary tables.

2) **X-Wing Hybrid KEM:** We have reserved wire-format identifiers for X-Wing hybrid KEM (wireId 0x0001, combining X25519 + ML-KEM-768). CryptoKit on Apple 26+ exposes HPKE cipher suites including X-Wing (ML-KEM-768 with X25519), motivating a standards-informed hybrid path. Our current implementation uses a domain-separated hybrid KEM composition (X25519 + ML-KEM-768) under wireId 0x0001 and reports on-device measurements in Table 4; full integration with CryptoKit's PQ-HPKE ciphersuite API remains future work.

3) **Key Rotation:** Session key rotation during long-lived connections is not addressed in the current design.

4) **Forward Secrecy for PQC suites:** v1 does not claim PFS for KEM-based PQC suites: ML-KEM encapsulation targets a peer's long-term KEM public key obtained during pairing. Accordingly, post-quantum confidentiality against store-now-decrypt-later adversaries holds only while long-term KEM private keys remain uncompromised.

5) **Cross-Platform Interoperability:** The wire format is documented but interoperability with non-Apple platforms requires additional validation.

TABLE 11

Simulated network impairment results (N=10,000 per condition, max retries=2). Completion and latency ranges span classic/liboqs/CryptoKit suites.

Condition (loss, RTT, jitter/reorder)	Completion rate (min)	p95 latency (ms, range)	handshakeFailed events / 10k (max)
mild (1%, 50 ms, ±20 ms)	1.0000	251–253	0
moderate (3%, 100 ms, ±50 ms)	1.0000	529–532	0
severe (5%, 200 ms, ±100 ms)	0.9989	1031–1033	11
reorder (10% reorder, 50 ms, ±20 ms)	1.0000	368–370	0

6) **Network Condition Evaluation:** We provide simulated impairment results in Section 7.5, but have not yet measured full end-to-end behavior under real NAT traversal, mobility, and heterogeneous access networks.

8.2 Future Directions

1) **Android PQC Integration:** Extend the CryptoProvider architecture to support Android's BouncyCastle PQC implementations.

2) **PFS Upgrade Path for PQC:** Move from static-recipient KEM to an ephemeral-ephemeral contribution (e.g., HPKE with an ephemeral sender key) or a hybrid design that provides at least classic PFS (e.g., X25519+ML-KEM) while preserving the existing provider/policy split and audit-event semantics.

3) **Tamper-Evident Telemetry:** Upgrade best-effort audit logs with cryptographic tamper evidence, e.g., per-session event hash-chains anchored into the handshake transcript or a post-handshake, cross-signed event digest exchanged between peers.

4) **Traffic Analysis Mitigations:** Reduce two-attempt identifiability via optional padding or constant-size framing, while preserving strictPQC semantics for deployments that require zero-downgrade behavior.

5) **Real-Network End-to-End Evaluation:** Measure handshake distributions across diverse access networks (same Wi-Fi, different NAT paths, LTE hotspots) and compare TCP vs QUIC control channels for 12 kB-class handshakes, validating that observed loss/jitter/reorder traces align with our fault taxonomy.

6) **Formal Verification:** Apply model checking to the handshake state machine to prove absence of deadlocks and race conditions.

7) **Performance Optimization:** Profile ML-KEM-768 encapsulation latency and explore hardware acceleration opportunities.

8) **Certificate-Based Identity:** Integrate with device certificates for enterprise deployment scenarios.

9 CONCLUSION

SkyBridge Compass shows that cryptographic agility and post-quantum readiness are attainable in P2P systems without sacrificing API stability or operational transparency. The implementation is production-focused. The layered CryptoProvider architecture enables adoption of new primitives as platforms evolve. The actor-isolated handshake state machine reduces concurrency hazards and limits sensitive-material exposure.

Our Apple-platform deployment indicates that native PQC APIs can be integrated with modest overhead when available, with graceful fallback to library-based or classic implementations on older systems. The structured security event model is designed to make cryptographic decisions auditable, supporting both real-time monitoring and post-incident analysis.

SkyBridge Compass prioritizes deterministic semantics and auditability in cryptographic migration. The default policy favors availability and explicit, observable downgrade signals, at the cost of attempt identifiability on the wire; strictPQC disables fallback for deployments that require zero-downgrade behavior and stronger capability privacy.

As post-quantum cryptography transitions from standardization to deployment, systems like SkyBridge Compass provide a practical template for managing this transition while maintaining security and usability across heterogeneous device fleets.

The limitations noted in Section 8 do not change the core contribution: an auditable migration contract with concurrency-safe handshake semantics that are reproducible from released artifacts under our stated threat model and measurement setup.

DATA AND ARTIFACT AVAILABILITY

Artifact release:

- URL: https://github.com/billlza/Skybridge-Compass
- Tag: `artifact-v2`
- Commit: `8635bf439cd4`
- Source archive: `artifact-v2.zip`
  SHA256=`b08659bf529c6295bf411bbe9f6150fa2ad555b437f072ce73fdce7a0d130903`
- Source archive: `artifact-v2.tar.gz`
  SHA256=`8890b67488194e7717bc10d207f85a77ef89d6c1ec00567222dea641b6ef7f73a`

See the repository README for environment details and CSV schemas.

CONFLICT OF INTEREST

The author declares no competing interests.

REFERENCES

[1] Bluetooth SIG, "Bluetooth Core Specification v5.4," 2023.
[2] Apple Inc., "Continuity," Apple Platform Security Guide, 2024.
[3] T. Taubert and C. A. Wood, "SPAKE2+, an Augmented PAKE," RFC 9383, IETF, Sept. 2023. [Online]. Available: https://datatracker.ietf.org/doc/rfc9383/
[4] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, IETF, 2018.

[5] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," RFC 9000, IETF, 2021. [Online]. Available: https://datatracker.ietf.org/doc/rfc9000/

[6] E. Barker and A. Roginsky, "Transitioning the Use of Cryptographic Algorithms and Key Lengths," NIST SP 800-131A Rev. 2, 2019.

[7] E. Barker et al., "Considerations for Achieving Cryptographic Agility: Strategies and Practices," NIST Cybersecurity White Paper (CSWP) 39, Dec. 2025, doi: 10.6028/NIST.CSWP.39.

[8] T. Perrin, "The Noise Protocol Framework," 2018. [Online]. Available: https://noiseprotocol.org/noise.pdf

[9] NIST, "Post-Quantum Cryptography Standardization," 2024. [Online]. Available: https://csrc.nist.gov/projects/post-quantum-cryptography. Accessed: Dec. 2025.

[10] Signal Foundation, "PQXDH Key Agreement Protocol," Signal Technical Documentation, 2023.

[11] Cloudflare, "Post-Quantum Cryptography Goes GA," Cloudflare Blog, Sept. 29, 2023. [Online]. Available: https://blog.cloudflare.com/post-quantum-cryptography-ga/. Accessed: Dec. 2025.

[12] P. Schwabe, D. Stebila, and T. Wiggers, "Post-Quantum TLS Without Handshake Signatures," in Proc. ACM CCS 2020, pp. 1461–1480.

[13] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe, "Post-Quantum Key Exchange—A New Hope," in Proc. USENIX Security 2016, pp. 327–343.

[14] J. Bos et al., "CRYSTALS—Kyber: A CCA-Secure Module-Lattice-Based KEM," in Proc. IEEE EuroS&P 2018, pp. 353–367.

[15] L. Ducas et al., "CRYSTALS—Dilithium: A Lattice-Based Digital Signature Scheme," IACR Trans. CHES, vol. 2018, no. 1, pp. 238–268.

[16] Apple Inc., "Apple Platform Security," Apple Security Guide, May 2024. [Online]. Available: https://support.apple.com/guide/security/

[17] D. Beyer et al., "Software Model Checking," in Handbook of Model Checking, Springer, 2018.

[18] B. Beurdouche et al., "A Messy State of the Union: Taming the Composite State Machines of TLS," in Proc. IEEE S&P 2015, pp. 535–552.

[19] Apple Inc., "Swift Concurrency," The Swift Programming Language, 2024.

[20] E. Rescorla and N. Modadugu, "Datagram Transport Layer Security Version 1.2," RFC 6347, IETF, Jan. 2012.

[21] M. Green and M. Smith, "Developers Are Not the Enemy!: The Need for Usable Security APIs," IEEE Security & Privacy, vol. 14, no. 5, pp. 40–46, Sept./Oct. 2016.

[22] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, and P.-Y. Strub, "Implementing TLS with Verified Cryptographic Security," in Proc. IEEE S&P 2013, pp. 445–459.

[23] Apple Inc., "CryptoKit Framework," Apple Developer Documentation, 2025. [Online]. Available: https://developer.apple.com/documentation/cryptokit. Accessed: Dec. 2025.

[24] NIST, "Module-Lattice-Based Digital Signature Standard," FIPS 204, Aug. 2024. [Online]. Available: https://csrc.nist.gov/pubs/fips/204/final

[25] D. McGrew, "Achieving Crypto Agility," in Proc. RSA Conference, 2019.

[26] R. Barnes, K. Bhargavan, B. Lipp, and C. Wood, "Hybrid Public Key Encryption," RFC 9180, IETF, Feb. 2022. [Online]. Available: https://datatracker.ietf.org/doc/rfc9180/

[27] Apple Inc., "Get ahead with quantum-secure cryptography," WWDC 2025 Session 314, June 2025. [Online]. Available: https://developer.apple.com/videos/play/wwdc2025/314/. Accessed: Dec. 2025.

[28] NIST, "Module-Lattice-Based Key-Encapsulation Mechanism Standard," FIPS 203, Aug. 2024. [Online]. Available: https://csrc.nist.gov/pubs/fips/203/final

[29] M. Barbosa, D. Connolly, J. Duarte, A. Kaiser, P. Schwabe, K. Varber, and B. Westerbaan, "X-Wing: The Hybrid KEM You've Been Looking For," IETF Internet-Draft, 2024. [Online]. Available: https://datatracker.ietf.org/doc/draft-connolly-cfrg-xwing-kem/

[30] D. J. Bernstein, "Curve25519: New Diffie-Hellman Speed Records," in Proc. PKC 2006, LNCS 3958, Springer, 2006, pp. 207–228.

[31] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-Speed High-Security Signatures," J. Cryptographic Engineering, vol. 2, no. 2, pp. 77–89, 2012.

[32] H. Krawczyk, "SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols," in Proc. CRYPTO 2003, LNCS 2729, pp. 400–425.

[33] H. Krawczyk and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)," RFC 5869, IETF, May 2010.

[34] M. Dworkin, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," NIST SP 800-38D, Nov. 2007.

[35] N. Aviram et al., "DROWN: Breaking TLS Using SSLv2," in Proc. USENIX Security 2016, pp. 689–706.

[36] D. Adrian et al., "Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice," in Proc. ACM CCS 2015, pp. 5–17.

[37] C. Meyer and J. Schwenk, "Lessons Learned From Previous SSL/TLS Attacks—A Brief Chronology of Attacks and Weaknesses," IACR Cryptology ePrint Archive, Report 2013/049, 2013.

[38] T. Tarman, R. Hutchinson, L. Pierson, P. Sholander, and E. Witzke, "Algorithm-Agile Encryption in ATM Networks," IEEE Computer, vol. 31, no. 9, pp. 57–64, Sept. 1998.

[39] B. Sullivan and V. Liu, "Web Application Security: A Beginner's Guide," McGraw-Hill, 2011, ch. 6 (Cryptographic Agility).