SkyBridge Compass: Supplementary Materials

Zi'ang Li, Peng Liu
Independent Researcher, Tianjin, China
E-mail: 2403871950@qq.com
Independent Researcher, Qiqihar, China

Supplementary Tables

This document provides extended data tables referenced in the main paper. All data is derived from the artifact repository CSV files.

Correspondence with Main Text:

- Table S1 (Latency): Referenced in main paper Section 7, supports Table 8 and Fig. 8
- Table S2 (RTT): Referenced in main paper Section 7, supports Table 8
- Table S3 (Message Sizes): Referenced in main paper Section 7, supports Table 8 and Fig. 9
- Table S7 (Loopback Wire Sizes): Supports Table 1 (baseline comparison)
- Tables S8–S9 (Repeatability): Supports Section 7. Tables report observed batch count B and (when $B \geq 2$) 95% confidence intervals across batches; set `SKYBRIDGE_BENCH_BATCHES=3–5` to reproduce multi-batch CIs.
- Table S4 (Traffic Padding): Supports Section 7 (traffic analysis mitigation quantization/overhead)
- Table S6 (Real-network micro-study): STUN path + 12 kB-class payload measurements across Wi-Fi / hotspot labels (generated from `Artifacts/realnet_*`). (Cross-NAT inbound-reachability depends on ISP/upstream equipment and may be infeasible without administrative control; see note below.)

Table S1: Full Handshake Latency Statistics

Table S2: Full RTT Statistics

Table S3: Message Size Breakdown by Field

Table S4: SBP2 Traffic Padding Quantization Summary

Table S5: SBP2 Bucket-Cap Sensitivity Study

Table S6: Real-Network Micro-Study (External Validity)

Note: Cross-NAT direct inbound tests (no overlay/relay) are not universally feasible. In particular, double-NAT topologies (ISP router/modem upstream) and IPv6 inbound filtering can prevent any inbound reachability even when devices have global IPv6 addresses. In such environments, all client samples time out with empty connect-time fields, and we omit those rows from the main table to avoid conflating "connectivity unavailable" with protocol performance.

Wire Size Summary:

- Classic: $293 + 318 + 2 \times 38 = 687$ B
- PQC: $6507 + 5419 + 2 \times 38 = 12{,}002$ B

Table S7: Loopback Wire-Size Baselines

Values are cert-dependent (localhost certificate). Noise-XX wire size is pattern-dependent and reflects the XX pattern used in this baseline. SkyBridge loopback wire sizes include framing and transport overhead; payload-only sizes are reported in Table S3.

Table S8: Repeatability Across Batches (Latency)

Table S9: Repeatability Across Batches (RTT)

Interpretation note: CryptoKit PQC RTT exhibits higher *between-batch* variance than Classic/liboqs in our environment (Apple Silicon, macOS 26.x), so its across-batch 95% CI can be substantially wider even when each batch uses N=1000 iterations. This reflects run-to-run scheduler/load sensitivity rather than an arithmetic error; raw batch values are directly recorded in `Artifacts/handshake_rtt_<date>.csv`.

Data Sources

All data is generated from the artifact CSV files:

- `Artifacts/handshake_bench_<date>.csv` (Tables S1, S8)
- `Artifacts/handshake_rtt_<date>.csv` (Tables S2, S9)
- `Artifacts/message_sizes_<date>.csv` (Table S3)
- `Artifacts/traffic_padding_<date>.csv` (Table S4)
- `Artifacts/traffic_padding_sensitivity_<date>.csv` (Table S5)
- `Artifacts/baseline_summary_<timestamp>.csv` (Table S7)

These files can be regenerated using `Scripts/run_paper_eval.sh` from the artifact repository; `Scripts/make_tables.py` uses the main-paper-pinned `\artifactdate` (or an explicit `ARTIFACT_DATE`) to avoid accidentally mixing datasets. System-level impact artifacts

TABLE S1
Supplementary Table S1: Full Handshake Latency Statistics.

Configuration	N	mean (ms)	std (ms)	p50 (ms)	p95 (ms)	p99 (ms)
Classic	1000	1.888	0.125	1.828	2.177	2.321
liboqs PQC	1000	3.470	0.363	3.404	4.182	4.601
CryptoKit PQC	1000	5.199	0.468	5.099	6.160	6.620

TABLE S2
Supplementary Table S2: Full RTT Statistics.

Configuration	N	mean (ms)	p50 (ms)	p95 (ms)	p99 (ms)
Classic	1000	0.541	0.531	0.574	0.624
liboqs PQC	1000	1.584	1.514	2.057	2.401
CryptoKit PQC	1000	2.452	2.364	3.047	3.447

TABLE S3
Supplementary Table S3: Message Size Breakdown by Field.

Message	Total (B)	Signature (B)	KeyShare (B)	Identity (B)	Overhead (B)
MessageA.Classic	293	64	32	36	161
MessageB.Classic	318	64	32	36	186
MessageA.PQC	6507	3309	1088	1956	154
MessageB.PQC	5419	3309	0	1956	154
Finished	38	0	0	0	38

TABLE S4
Supplementary Table S4: SBP2 traffic padding quantization summary. "raw"/"padded" are aggregate bytes across events; overhead is relative to raw. Top bucket reports the most frequent bucket size (share).

Label	wraps	unwraps	raw (B)	padded (B)	overhead (%)	top bucket
rx	0	56	158110	305152	93%	256B (36%)
CP/fileTransferRequest	4	0	1460	2048	40%	512B (100%)
CP/heartbeat	3	0	168	768	357%	256B (100%)
CP/systemCommand	5	0	1055	1280	21%	256B (100%)
DP/32B	2	0	136	512	276%	256B (100%)
DP/300B	5	0	1680	2560	52%	512B (100%)
DP/900B	2	0	1872	2048	9%	1024B (100%)
DP/1400B	2	0	2872	4096	43%	2048B (100%)
DP/4KiB	1	0	4132	8192	98%	8192B (100%)
DP/16KiB	4	0	65680	131072	100%	32768B (100%)

TABLE S5
Supplementary Table S5: SBP2 bucket-cap sensitivity study. We vary the maximum bucket size (cap) and report padding overhead, cap coverage (fraction of frames whose framed payload exceeds the cap), and a privacy proxy (bucket entropy) for representative handshake, control, and data-plane workloads.

Label	64 KiB cap			128 KiB cap			256 KiB cap		
	overhead (%)	>cap (%)	entropy (b)	overhead (%)	>cap (%)	entropy (b)	overhead (%)	>cap (%)	entropy (b)
HS/MessageA	-	-	-	-	-	-	-	-	-
HS/MessageB	-	-	-	-	-	-	-	-	-
HS/Finished	-	-	-	-	-	-	-	-	-
CP/heartbeat	357%	0%	0.00	357%	0%	0.00	357%	0%	0.00
CP/systemCommand	21%	0%	0.00	21%	0%	0.00	21%	0%	0.00
CP/fileTransferRequest	40%	0%	0.00	40%	0%	0.00	40%	0%	0.00
DP/32B	-	-	-	276%	0%	0.00	276%	0%	0.00
DP/300B	52%	0%	0.00	52%	0%	0.00	52%	0%	0.00
DP/900B	9%	0%	0.00	9%	0%	0.00	9%	0%	0.00
DP/1400B	43%	0%	0.00	43%	0%	0.00	43%	0%	0.00
DP/4KiB	-	-	-	98%	0%	0.00	98%	0%	0.00
DP/16KiB	100%	0%	0.00	100%	0%	0.00	100%	0%	0.00
DP/rdpMix	0%	100%	1.50	14%	0%	1.00	35%	0%	1.50
DP/fileMix	0%	100%	1.50	0%	100%	1.50	33%	0%	0.81

(`system_impact_...`) are generated by a separate benchmark suite and are pinned consistently in the main paper via `\artifactdateSystemImpact`.

Artifact release:

TABLE S6

Supplementary Table S6: Real-network micro-study (STUN path + TCP payload). Each row is one network condition label. STUN metrics capture path RTT/loss and a conservative NAT classification. E2E metrics report success rate and p50 completion time for two payload sizes (classic 687 B vs PQC 12,002 B), along with the delta (PQC minus classic). Failure taxonomy is summarized for the PQC size as timeout rate and other failure rate.

Label	IFace	Exp	Con	NAT	STUN loss	STUN RTT p50/p95 (ms)	ok_c	$p50_c$ (ms)	ok_p	$p50_p$ (ms)	Δp50 (ms)	PQC fail (to/other)
home_wifi	wifi	0	0	unknown	0.0000	64.206/138.442	1.0000	9.821	1.0000	16.296	6.475	0.0000/0.0000
phone_hotspot_5ga_r18	wifi	1	1	unknown	0.0000	103.018/175.414	1.0000	19.511	1.0000	27.588	8.077	0.0000/0.0000

TABLE S7

Loopback wire-size baselines from pcap capture (N=1000). Wire bytes count full packet lengths observed on the loopback interface; TLS/QUIC/DTLS values are cert-dependent.

Protocol	Wire p50 (B)	Wire p95 (B)
TLS 1.3	8,824	8,948
QUIC	7,830	14,427
WebRTC DTLS	3,090	3,162
Noise XX	2,560	2,672
SkyBridge (classic)	4,874	5,028
SkyBridge (liboqs)	37,779	44,609
SkyBridge (CryptoKit)	19,009	19,257

- URL: https://github.com/billlza/Skybridge-Compass
- Tag: `artifact-v3`
- Commit: `c23a8b4a3d01`
- Source archive: `artifact-v3.zip` SHA256=`c370f07da6fe825c2132f447db3287e0689d0344b26b4d97ff4f043d2cbac1e3`
- Source archive: `artifact-v3.tar.gz` SHA256=`ff467cdc761a9a6528de871f0fd8663e788e0aa7a6af5b8883199a2be68642c9`

Supplementary Methods and Details

Wire Format and Validation Rules

Key Share Semantics: The `keyShares[].shareBytes` field has suite-dependent interpretation:

- **DH suites (X25519):** `shareBytes` = ephemeral public key (32 bytes)
- **KEM suites (ML-KEM-768):** `shareBytes` = encapsulated key / ciphertext (`enc`, 1088 bytes)

This distinction matters: for DH, the Responder uses the Initiator's public key to compute a shared secret; for KEM, the Initiator encapsulates to the Responder's long-term KEM public key (obtained during pairing), and `shareBytes` carries the resulting `enc`. The Responder decapsulates using their private key.

Forward Secrecy Note: This is a static KEM exchange to a long-term KEM public key, and thus does *not* provide traditional PFS semantics. If the long-term KEM private key is compromised later, a passive attacker who recorded ciphertexts may be able to recover past session keys.

Nonce Freshness: Each party contributes a 32-byte nonce (`clientNonce` in A, `serverNonce` in B). Both are bound into the KDF info parameter, ensuring symmetric freshness and enabling a unique session identifier:

```
handshakeId = SHA256(replayTag ||
initiatorNonce || responderNonce ||
suiteWireIdLE)
```

To prevent short-window replay attacks, implementations SHOULD cache recent `handshakeId` values (or the `(initiatorNonce, responderNonce)` pair) and reject duplicates within a configurable window (default: 5 minutes).

Key Share Binding:

- The `keyShares[]` array contains at most two entries to bound message size. Under the two-attempt strategy, each `MessageA` is homogeneous (PQC/hybrid-only or classic-only), so the two entries (if present) belong to the same suite group for that attempt.
- Each entry is a `(suiteId, shareBytes)` tuple.
- The Responder MUST select a suite for which the Initiator provided a key share; otherwise reject with `missingKeyShare`.
- This binds negotiation to actual cryptographic material, preventing the "TLS key_share mismatch" class of bugs.

Explicit Key Confirmation (Finished Frames):

- The Responder sends a short `Finished_R2I` frame authenticated under the newly derived session keys. The Initiator verifies it and replies with `Finished_I2R`.
- A session is established only after both Finished frames are verified, reducing responder-side half-open state under failures.
- Finished frames are fixed-size authenticated messages (38 bytes each: 4-byte magic, 1-byte version, 1-byte direction, 32-byte HMAC), adding negligible wire overhead compared to PQC payloads.
- Finished MACs are computed over the handshake transcript using per-direction keys:

```
finishedMAC
= HMAC-SHA256(finishedKey,
transcriptHash)
finishedKey_R2I = HKDF(sessionKey,
  info="SkyBridge-FINISHED|R2I|")
finishedKey_I2R = HKDF(sessionKey,
  info="SkyBridge-FINISHED|I2R|")
```

Anti-Downgrade Invariant:

- The Initiator MUST verify that `selectedSuite` is a member of `supportedSuites[]` it originally sent.
- It MUST also confirm that `keyShares[]` contains an entry for `selectedSuite`.
- Since `sigB` commits to `MessageA` via `transcriptA`, it binds the initiator's proposal.

TABLE S8

Supplementary Table S8: Repeatability across independent benchmark batches (latency). Table reports observed batch count B. Cells report mean and (when $B \geq 2$) $\pm$ 95% CI across batches; each batch uses N=1000 iterations after 10 warmup runs.

Configuration	B	N/batch	mean (ms)	p50 (ms)	p95 (ms)
Classic	3	1000	1.869 ± 0.041	1.828 ± 0.001	2.067 ± 0.237
liboqs PQC	3	1000	3.481 ± 0.106	3.410 ± 0.096	4.196 ± 0.297
CryptoKit PQC	3	1000	5.248 ± 0.310	5.147 ± 0.277	6.148 ± 0.389

TABLE S9

Supplementary Table S9: Repeatability across independent benchmark batches (RTT). Table reports observed batch count B. Cells report mean and (when $B \geq 2$) $\pm$ 95% CI across batches; each batch uses N=1000 iterations after 10 warmup runs.

Configuration	B	N/batch	mean (ms)	p50 (ms)	p95 (ms)
Classic	3	1000	0.539 ± 0.011	0.531 ± 0.006	0.571 ± 0.021
liboqs PQC	3	1000	1.588 ± 0.027	1.521 ± 0.022	2.073 ± 0.106
CryptoKit PQC	3	1000	2.658 ± 1.190	2.570 ± 1.188	3.255 ± 1.204

- Any tampering with the initiator's offered suites or key shares will cause `sigB` verification to fail.

Canonical Encoding Rules (V1 Wire Format):

- `supportedSuites[]`: preference order, signed as-is (first = most preferred)
- `keyShares[]`: entries follow the suite preference order; only suites with provided shares appear
- All lists use 2-byte little-endian length prefix
- All integers use little-endian encoding
- Canonical encoding is byte-for-byte specified. Implementations MUST NOT reserialize with language-native encoders (e.g., JSON, PropertyList), as this may introduce non-determinism

Endianness rationale: V1 wire format uses little-endian to align with platform conventions (ARM64 native order). The V2 TLV format uses network byte order (big-endian) for TLV length fields to align with TLV conventions in IETF protocols. The two formats serve different purposes: V1 is the on-wire message encoding and is used as transcript input in v1 deployments (byte-for-byte deterministic). V2 TLV is an optional transcript-hashing format for forward compatibility and is exercised by regression tests; it is not used on the wire in v1.

Note: The 32-byte `clientNonce/serverNonce` in message fields are distinct from the 12-byte `aeadNonce` used internally by AES-GCM for authenticated encryption.

Suite Identifiers and 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. Algorithm suite identifiers use a structured 16-bit wire format enabling forward compatibility:

Note: X-Wing is a hybrid KEM combining X25519 + ML-KEM-768; it does not include a signature algorithm. The suite identifier specifies the full primitive set, with ML-DSA-65 as the preferred signature algorithm when available. In v1, protocol signatures are bound to the suite group: ML-DSA-65 for PQC/hybrid suites and Ed25519 for classic suites. Secure Enclave P-256 ECDSA is used only for optional device proof-of-possession and is not used for the main protocol signatures (`sigA`/`sigB`).

Unknown suite identifiers are parsed as `unknown(wireId)` rather than causing parse failures. This allows older clients to gracefully reject unsupported suites during negotiation.

Transport/MTU considerations: handshake messages are length-prefixed and carried on reliable control channels (TCP or QUIC streams), so fragmentation is handled by the transport layer rather than by the handshake format itself. QUIC datagrams are reserved for latency-critical media frames; when datagrams are used for other payloads, senders must respect the maximum datagram size and chunk accordingly.

KEM-DEM Envelope

The sealed box structure encapsulates KEM-based authenticated encryption output with explicit DoS protection. Our construction follows the KEM-DEM (Key Encapsulation Mechanism + Data Encapsulation Mechanism) paradigm: KEM.Encapsulate() $\rightarrow$ HKDF-SHA256 $\rightarrow$ AES-256-GCM. We call this an "HPKE-inspired KEM-DEM envelope" to distinguish it from RFC 9180 HPKE, which includes additional features such as multiple modes (Base, PSK, Auth, AuthPSK), context exporters, and a more complex key schedule.

We model v1 as a constrained HPKE Base-mode instance: the KEM encapsulation yields a shared secret, HKDF-Extract/Expand derives an AEAD key/nonce with domain-separated info (role, suite ID, transcript hash), and AEAD provides confidentiality and ciphertext integrity under unique nonces. This makes the security dependencies explicit and isolates deviations from RFC 9180 to the simplified header/nonce/tag framing.

On Apple 26+ platforms, CryptoKit provides native HPKE with quantum-secure cipher suites including X-Wing (ML-KEM-768 + X25519). When available, implementations SHOULD use CryptoKit's HPKE API directly rather than this compatibility envelope.

Security Goals:

- **PQC suites:** ML-KEM provides IND-CCA2 KEM security (per FIPS 203).

TABLE S10
Message field validation rules.

Field	Validation	Failure Action
version	Must equal protocol version (1)	Reject with versionMismatch
supportedSuites	Must contain at least one suite supported by local implementation; unknown IDs are ignored for negotiation but still transcript-bound	Reject with suiteNegotiationFailed
keyShares	Unique suiteId per entry, max 2 entries, each shareBytes must match its suiteId's expected length	Reject with invalidMessageFormat
selectedSuite	Must be in supportedSuites AND have matching keyShare	Reject with missingKeyShare
clientNonce/serverNonce	Must be 32 bytes	Reject with invalidMessageFormat
sigA/sigB	Must verify against respective identityPubKey	Reject with signatureVerificationFailed

TABLE S11
Encoding scope (V1 vs V2).

Component	Encoding	Length endianness	Used in v1 deployment
`MessageA`/`MessageB` wire bytes	V1 deterministic	little-endian	yes
Transcript hash input (v1)	V1 deterministic	little-endian	yes
Transcript hash input (v2)	V2 TLV canonical	big-endian	no (experimental)

TABLE S12
Suite identifier ranges and components.

Range	Category	Suite Components	Examples
0x00xx	Hybrid PQC (preferred)	X-Wing KEM, ML-DSA-65, AES-256-GCM, HKDF-SHA256	0x0001
0x01xx	Pure PQC	ML-KEM-768, ML-DSA-65, AES-256-GCM, HKDF-SHA256	0x0101
0x10xx	Classic	X25519 DHKEM, Ed25519, AES-256-GCM, HKDF-SHA256	0x1001
0xF0xx	Experimental	Reserved for testing	-

TABLE S13
Post-quantum coverage by policy and established suite (v1). "PQC confidentiality" refers to resistance against store-now-decrypt-later adversaries *while long-term KEM private keys remain uncompromised*; v1 does not claim PFS for KEM-based PQC suites.

Policy	Outcome	KEM / key establishment	Protocol signature (`sigA`/`sigB`)	PQ conf.	PQ auth.	PFS
default	PQC established	ML-KEM-768 (static recipient key)	ML-DSA-65	yes	yes	no
default	classic fallback	X25519 ephemeral DH	Ed25519	no	no	yes
strictPQC	PQC established	ML-KEM-768 (static recipient key)	ML-DSA-65	yes	yes	no
strictPQC	PQC unavailable	– (handshake fails)	–	–	–	–

- **Classic suites:** X25519 ephemeral-DH encapsulation (DHKEM-style); security relies on the X25519/Gap-DH assumption and HKDF key separation.
- **Payload encryption:** INT-CTXT and IND-CPA via AES-256-GCM in v1 (compat KEM-DEM), and RFC 9180 HPKE AEAD in v2 (e.g., ChaCha20-Poly1305 in our classic provider).
- **Key separation:** HKDF with context-specific info parameters including role binding.

Not Covered (delegated to RFC 9180 HPKE on Apple 26+):

- Sender authentication modes (Auth, AuthPSK)
- Incremental AEAD for streaming

Used in our handshake (classic v2):

- HPKE exporter for deriving the per-session shared secret with explicit context binding

Format Versions:

The implementation supports two sealed box formats:

v1 (Compatibility KEM-DEM): Used when native HPKE is unavailable. Explicit nonce and tag fields.

Header (17 bytes):

magic	version	suite	flags
4B	1B	2B	2B
encLen	nonceLen	tagLen	ctLen
2B	1B	1B	4B

Body (v1):

encapsulatedKey	nonce	ciphertext	tag
encLen	12B	ctLen	16B

v2 (Native HPKE): Used with CryptoKit HPKE. Nonce and tag are embedded in the AEAD output.

Header (17 bytes):

magic	version	suite	flags
4B	1B	2B	2B
encLen	nonceLen	tagLen	ctLen
2B	0	0	4B

Body (v2):

encapsulatedKey	ciphertext (AEAD output)
encLen	ctLen (includes auth tag)

Version Detection: Parsers distinguish v1 from v2 by checking `nonceLen` and `tagLen`:

- v1: `nonceLen = 12`, `tagLen = 16`
- v2: `nonceLen = 0`, `tagLen = 0` (AEAD details encapsulated by library)

Length limits enforce DoS protection:

- `encLen <= 4096` bytes (sufficient for ML-KEM-768's 1088-byte ciphertext)
- v1: `nonceLen = 12` bytes, `tagLen = 16` bytes (AES-GCM fixed)
- v2: `nonceLen = 0`, `tagLen = 0` (embedded in ciphertext)
- `ctLen <= 64KB` (handshake phase, pre-authentication window)
- `ctLen <= 256KB` (post-authentication)

Parsing uses overflow-safe arithmetic and validates each field before allocation.

Platform Support Matrix

[1] ML-KEM-1024 and ML-DSA-87 are available in CryptoKit on Apple 26+ but not currently used by our implementation. X-Wing hybrid KEM (wireId 0x0001) is reserved for future use. Secure Enclave PQC keys are only available on Apple 26+; earlier versions fall back to software PQC and P-256 Secure Enclave PoP.

Native PQC Integration

The `ApplePQCCryptoProvider` wraps CryptoKit ML-KEM and ML-DSA APIs, enabling platform-backed PQC suites and HPKE-based encapsulation. This construction (KEM → HKDF → AEAD) is inspired by HPKE but does not implement the full RFC 9180 specification. On Apple 26+ platforms, CryptoKit exposes HPKE cipher suites including X-Wing (ML-KEM-768 with X25519), enabling a standards-aligned replacement for our compatibility envelope. This layer can then be replaced with direct CryptoKit PQ-HPKE calls.

Conditional Compilation Strategy

The `HAS_APPLE_PQC_SDK` flag gates **all** PQC type references. Importantly, `@available` only controls *runtime* availability; it does not prevent *compile-time* failures when the SDK lacks the PQC symbols.

For reproducible builds across toolchains, we intentionally **do not** enable `HAS_APPLE_PQC_SDK` by default in SwiftPM (because `.when(platforms: [.macOS])` does not reflect SDK availability and will break older Xcode builds). Instead, projects inject the flag from build settings **only** when compiling with the Apple 26 SDK (Xcode 26+): `OTHER_SWIFT_FLAGS = $(inherited) -DHAS_APPLE_PQC_SDK`

This keeps the codebase buildable on older Xcode versions (classic provider path), while enabling native CryptoKit PQC providers only when the correct SDK is present.

Security Event Emission

All cryptographic decisions emit structured events. The `SecurityEventEmitter` actor implements backpressure with per-subscriber queues and meta-event rate limiting to prevent recursive overflow.

Secure Enclave Integration

For hardware-backed signing, the `SigningCallback` protocol enables Secure Enclave integration. The `HandshakeDriver` prioritizes callback-based signing over raw key material, ensuring private keys never leave the Secure Enclave.

Availability and fallback. Secure Enclave–backed ML-DSA/ML-KEM keys are only available on macOS 26+ via CryptoKit. On macOS 14–15, PQC operations fall back to liboqs (software), and Secure Enclave is used only for P-256 ECDSA proof-of-possession keys. When Secure Enclave PQC is used, the implementation relies on CryptoKit key types `SecureEnclave.MLDSA*` and `SecureEnclave.MLKEM*`, which are gated by the macOS 26+ SDK and runtime availability checks.

Signing Key Hierarchy

The system supports two complementary signing mechanisms:

1) **CryptoProvider signing (Protocol Signature):** Uses the active suite's signature algorithm (Ed25519 for classic, ML-DSA-65 for PQC) for protocol-level identity verification. Keys are managed by the CryptoProvider and stored in software. This is the primary signature used in `sigA/sigB` fields of handshake messages.
2) **Secure Enclave signing (Device PoP):** Uses EC P-256 with ECDSA via `SecureEnclave SigningCallback` to prove the peer controls a key stored in Secure Enclave. Private keys never leave the Secure Enclave hardware. Note: This provides proof-of-possession of a hardware-backed key, not full device attestation (Apple does not expose a general-purpose attestation API with certificate chains at the application layer). The security value derives from the key being pinned during initial pairing.

Implementation Note. `DeviceIdentityKeyManager` creates P-256 keys in Secure Enclave (when available) for hardware-backed device identity. These keys are used for the optional `seSigA/seSigB` proof-of-possession signatures, NOT for the primary protocol signatures (`sigA/sigB`). The primary protocol signatures use Ed25519

TABLE S14
Platform support matrix.

Platform	PQC Provider	Algorithms	Notes
macOS 26+	ApplePQCProvider	ML-KEM-768, ML-KEM-1024[1], ML-DSA-65, ML-DSA-87[1]	Native CryptoKit
macOS 14–15	OQSPQCProvider	ML-KEM-768, ML-DSA-65	liboqs fallback
macOS 14–15	ClassicProvider	X25519, Ed25519	If liboqs unavailable
iOS 26+	ApplePQCProvider	ML-KEM-768, ML-KEM-1024[1], ML-DSA-65, ML-DSA-87[1]	Native CryptoKit
iOS 17–18	ClassicProvider	X25519, Ed25519	liboqs not bundled

(classic) or ML-DSA-65 (PQC) keys generated by the CryptoProvider.

The `FallbackSigningCallback` provides automatic fallback from Secure Enclave to CryptoProvider when hardware signing is unavailable (e.g., on devices without Secure Enclave or when the key has not been provisioned).

Use Case Separation:

- CryptoProvider (Ed25519/ML-DSA): Primary protocol signatures, cross-platform interoperability, suite-negotiated
- Secure Enclave (P-256 ECDSA): Optional hardware-backed proof-of-possession, proving control of a non-exportable key

Both mechanisms can coexist in a single handshake: CryptoProvider for primary protocol signatures (`sigA`/`sigB`), Secure Enclave for optional hardware-backed proof-of-possession (`seSigA`/`seSigB`).

Signature Verification Rules:

- Primary signatures (`sigA`/`sigB`) are mandatory and must verify against the peer's `identityPubKey`.
- For paired peers, `identityPubKey` must match the pinned value from initial pairing.
- Optional SE signatures (`seSigA`/`seSigB`) elevate trust when present and valid.
- Verification order: primary signature, identity pinning, optional SE signature.

Pre-Negotiation Signature Selection and Two-Attempt Strategy

A fundamental challenge in our protocol is the "chicken-and-egg" problem: `sigA` must be generated *before* suite negotiation completes, yet the signature algorithm should be consistent with the negotiated suite. We resolve this through pre-negotiation signature selection and a two-attempt strategy.

Pre-Negotiation Signature Selection: The signature algorithm for `sigA` is determined by the `offeredSuites` in MessageA, not by the final `selectedSuite`. The `PreNegotiationSignatureSelector` builds offered suites based on the attempt strategy and selects the appropriate signature algorithm (ML-DSA-65 for PQC suites, Ed25519 for classic).

Homogeneity Invariant: Each attempt's `offeredSuites` must be homogeneous with respect to `sigAAlgorithm`: ML-DSA-65 requires all PQC suites; Ed25519 requires all classic suites. This invariant is enforced

TABLE S15
ML-DSA-65 Key Sizes (FIPS 204 Compliance).

Component	Expected	Measured	Status
Public Key	1952 B	1952 B	OK
Secret Key	4032 B	4032 B	OK
Signature	3309 B	3309 B	OK

at compile-time through the type system and at runtime through `HandshakeDriver` initialization validation.

Two-Attempt Strategy: To support interoperability between PQC-capable and classic-only devices, we employ a two-attempt strategy: (1) PQC attempt with ML-DSA-65 signatures, then (2) classic fallback with Ed25519 if the PQC attempt fails due to provider unavailability or suite negotiation failure. In particular, for a classic-only peer without a pinned PQC KEM public key from pairing, the PQC attempt fails locally during `MessageA` construction (no key shares $\rightarrow$ `suiteNegotiationFailed`), so fallback is immediate and is not driven by network timeout.

Fallback Security: Not all failures trigger fallback.

Allowed:

- `pqcProviderUnavailable`
- `suiteNotSupported`
- `suiteNegotiationFailed`

Blocked:

- `timeout`
- `signatureVerificationFailed`
- `identityMismatch`
- `replayDetected`

Timeout-based fallback is explicitly blocked to prevent attackers from forcing downgrade through packet dropping. Per-peer fallback is rate-limited (default 5-minute cooldown) to prevent rapid downgrade cycling.

ML-DSA Key Sizes

Key Size Reference

[1] Apple CryptoKit uses a seed-based compact format for private key serialization (`integrityCheckedRepresentation`). This is more storage-efficient than the FIPS 203/204 expanded format. Public keys use `rawRepresentation` which matches FIPS standard sizes. Measurements performed on macOS 26.0 (Tahoe) SDK.

[2] Not measured directly; projected from the seed-based pattern observed in ML-KEM-768/ML-DSA-65 (ratio of

TABLE S16
Key size reference.

Algorithm	Public Key	Private Key (Apple)[1]	Private Key (FIPS)	Ciphertext/Signature
ML-KEM-768	1184 B	96 B	2400 B	1088 B
ML-KEM-1024	1568 B	128 B[2]	3168 B	1568 B
ML-DSA-65	1952 B	64 B	4032 B	3309 B
ML-DSA-87	2592 B	96 B[2]	4896 B	4627 B
X25519	32 B	32 B	-	32 B
Ed25519	32 B	32 B	-	64 B

FIPS expanded size to Apple compact size). These algorithms are available in CryptoKit but not exercised by our current benchmark suite.

Security Event Types

X-Wing Wire Size (Measured)

X-Wing is a hybrid KEM combining X25519 + ML-KEM-768. In our v1 handshake, the initiator carries only a KEM ciphertext in MessageA; thus X-Wing increases the MessageA keyshare from 1088 B (ML-KEM-768) to 1120 B (X25519 32 B + ML-KEM-768 1088 B), while MessageB remains keyshare-free. We measure X-Wing directly via the native CryptoKit provider (wireId 0x0001) and report per-field sizes in Supplementary Table S3 (`MessageA.XWing/MessageB.XWing`). The measured total handshake size (MessageA + MessageB + 2×Finished) is **12,195 B**.

Reproducibility

Test Commands. One-shot evaluation: `Scripts/run_paper_eval.sh`. Individual suites: `swift test --filter TestName`. Representative suite names:

- `HandshakeBenchmarkTests`
- `HandshakeFaultInjectionBenchTests`
- `PolicyDowngradeBenchTests`
- `MessageSizeSnapshotTests`

All CSV outputs are date-stamped under `Artifacts/`. To prevent mixing datasets across runs, set a fixed date suffix: `ARTIFACT_DATE=YYYY-MM-DD` (or `SKYBRIDGE_ARTIFACT_DATE`).

Repeatability (Multi-batch). Repeatability tables report observed batch count B and (when $B \geq 2$) mean $\pm$ 95% confidence intervals across independent batches. To reproduce multi-batch CIs, rerun with process restarts: `ARTIFACT_DATE=YYYY-MM-DD SKYBRIDGE_BENCH_BATCHES=3--5 bash Scripts/run_paper_eval.sh`.

Real-network micro-study (External validity). We provide two lightweight scripts: (i) `Scripts/run_real_network_probe.swift` (STUN RTT + conservative NAT classification), and (ii) `Scripts/run_real_network_e2e.swift` (TCP connect + first-byte + completion timing for a 12 kB-class payload). Pin filenames with `ARTIFACT_DATE=YYYY-MM-DD`, run them across labels (same Wi-Fi / cross-NAT / hotspot), then aggregate into a Supplementary table via: `python3 Scripts/aggregate_realnet.py`.

Note on inbound reachability. Some home networks do not provide an inbound-reachable IPv4 address (e.g., double NAT behind an ISP router, CGNAT, DS-Lite, or WAN IPv4 shown as `0.0.0.0`). In these cases, IPv4 port-forwarding experiments will fail (client samples report `timeout` with empty `connect_ms`). For a true cross-NAT condition without modifying upstream equipment, prefer IPv6 direct connectivity (when available) with an explicit IPv6 firewall allow-rule for TCP `44444`. If only an overlay/relay path is feasible, label the condition accordingly (e.g., `cross_nat_via_relay`) and interpret results as overlay-mediated connectivity rather than raw Internet inbound reachability.

Regression Testing

Transcript integrity uses V1 (deterministic) and V2 (TLV canonical) encoding formats, validated with 14 test cases (100% pass rate). TLV encoding uses 1-byte tags, 4-byte big-endian length fields, and variable-length values. The complete regression matrix validates Requirements 12.1–12.6; all 15 regression tests pass, and all 3 compile-fail harness tests correctly produce compile errors.

Key Properties: (1) P-256 cannot be used as a protocol signature algorithm (compile-time enforced); (2) PQC attempts use only PQC suites, classic attempts use only classic suites; (3) Only PQC-unavailability errors trigger fallback; (4) Legacy P-256 requires authenticated channel or existing TrustRecord; (5) All fallback and legacy acceptance events include full audit context.

Artifact Map

Table S18 maps implementation components to source files in the artifact repository (https://github.com/billlza/Skybridge-Compass, tag=`artifact-v3`).

TABLE S17
Security event types.

Event Type	Severity	Trigger
`cryptoProviderSelected`	info/warning	Provider factory selection
`cryptoDowngrade`	warning	PQC to classic fallback
`handshakeFailed`	warning	Any handshake failure
`signatureVerificationFailed`	high	Invalid peer protoSignature
`secureEnclaveVerificationFailed`	warning	Invalid secureEnclaveSignature
`identityMismatch`	high	identityPubKey does not match pinned key
`contextZeroized`	info	Sensitive material cleared
`suiteNegotiationFailed`	warning	No common suite found
`unexpectedStateTransition`	high	Actor reentrancy detected

TABLE S18
Implementation artifact map.

Component	File (Lines)
Provider Protocol	`CryptoProviderProtocol.swift` (L24–110)
Handshake Types	`HandshakeTypes.swift` (L270–289)
Provider Factory	`CryptoProviderFactory.swift` (L18–139)
Handshake Driver	`HandshakeDriver.swift` (L88–166, L992–1012)
Handshake Context	`HandshakeContext.swift` (L27–100, L848–900)
Secure Bytes	`SecureBytes.swift` (L18–99)
Message Signatures	`HandshakeMessages.swift` (L555–567, L797–818)
Apple PQC Provider	`ApplePQCProvider.swift` (L22–120)
Security Events	`SecurityEventEmitter.swift` (L21–140)
SE Signing	`SecureEnclaveSigningCallback.swift` (L40–94)
Signature Selector	`PreNegotiationSignatureSelector.swift` (L69–96)