LP-9702: Z-Wing Hybrid Post-Quantum Secure Channel

Zach Kelling

Lux Industries

Abstract

Z-Wing is a hybrid post-quantum secure channel for Lux node-to-node and agent-to-agent communication. The key encapsulation mechanism is X-Wing (ML-KEM-768 + X25519) per `draft-connolly-cfrg-xwing-kem`, used verbatim with the standard SHA3-256 combiner so that Z-Wing is wire-compatible with any conformant X-Wing implementation. Identity binding is hybrid Ed25519 + ML-DSA-65 with both signatures committing to the same handshake transcript. The handshake is one round trip; after completion the channel exposes `net.Conn`, so any byte-stream protocol — in particular the ZAP wire format (`luxfi/api/zap`) — runs unmodified above it. Z-Wing supersedes LP-9701 (RNS hybrid-PQ link), which used a Lux-specific HKDF combiner and inlined identity binding into the handshake construction. LP-9702 collapses the two PQ surfaces in Lux today (LP-9701 in Go, `zap-protocol` hybrid in Rust) onto a single canonical construction.

Contents

1 Motivation

Two hybrid post-quantum constructions are deployed in Lux today.

1. **LP-9701 (RNS hybrid link, Go).** Implemented in `luxfi/node/network/dialer/rns_link.go`. Combines X25519 and ML-KEM-768 with an ad-hoc combiner: the ML-KEM contribution is derived deterministically from a sorted hash of both peers' ML-KEM ephemeral public keys (rather than a true KEM encapsulation), then mixed with the X25519 ECDH output via HKDF-SHA256 under Lux-specific labels (`RNS-HybridLink-v1`, `rns-hybrid-link-keys`, `RNS-MLKEM-Hybrid-v1`). Identity binding (Ed25519 + ML-DSA-65) is fused into the handshake messages.

2. **`zap-protocol` hybrid (Rust).** Implemented in `zap/src/crypto.rs`. X25519 + ML-KEM-768 with HKDF-SHA256 under the Lux-specific label `ZAP-HYBRID-HANDSHAKE-v1`. Identity is layered separately by the application.

Both constructions are sound but neither is the IETF X-Wing combiner (`draft-connolly-cfrg-xwing-kem`) which uses SHA3-256 with a fixed 6-byte label and a fixed input order. That makes cross-org and cross-language interop a per-implementation matter rather than a standards-conformant property.

LP-9702 collapses both onto a single canonical construction:

- KEM: X-Wing exactly as specified by the IETF draft.
- Identity: Ed25519 + ML-DSA-65, both signing the handshake transcript hash.
- Layering: the channel is a `net.Conn`; any byte-stream protocol rides on top with no further specification.

2 Architecture

2.1 Layering

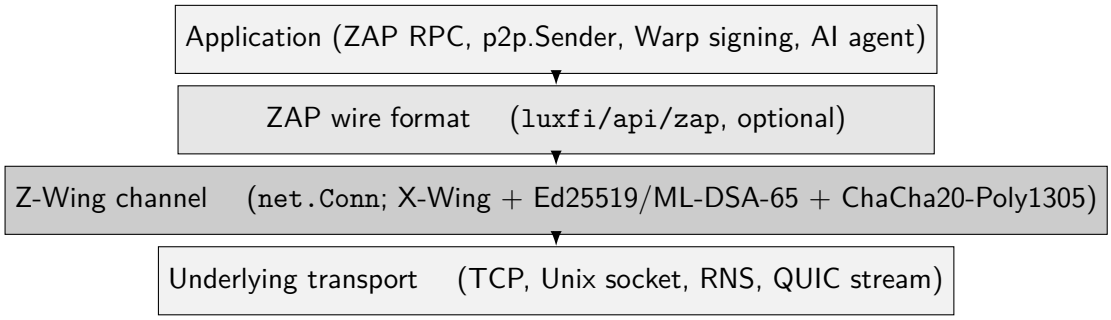

Figure 1: Z-Wing layering. The Z-Wing channel is a `net.Conn` whose underlying byte stream is any reliable, ordered transport. ZAP and any other application protocol ride above without modification.

2.2 Components

Component	Construction
KEM	X-Wing = ML-KEM-768 (FIPS 203) + X25519 (RFC 7748), combined with SHA3-256
Identity	Ed25519 (RFC 8032) + ML-DSA-65 (FIPS 204), parallel signatures over transcript hash
KDF	HKDF-SHA3-256 (RFC 5869 with FIPS 202 hash)
AEAD	ChaCha20-Poly1305 (RFC 8439)
Transcript	SHA3-256 (FIPS 202)

2.3 Channel interface

The Z-Wing channel implements the Go `net.Conn` contract: `Read`, `Write`, `Close`, `LocalAddr`, `RemoteAddr`, the three deadline setters. After `Handshake` returns, the channel is authenticated, confidential, integrity-protected, and forward-secret. `Read` and `Write` operate on plaintext bytes; encryption and framing are handled internally.

3 KEM Construction (Normative)

3.1 Component primitives

Quantity	Symbol	Size (bytes)
ML-KEM-768 encapsulation key	pk_M	1184
ML-KEM-768 decapsulation key	sk_M	2400
ML-KEM-768 ciphertext	ct_M	1088
ML-KEM-768 shared secret	ss_M	32
X25519 public key	pk_X	32
X25519 private key	sk_X	32
X25519 ephemeral public key (sender)	ct_X	32
X25519 shared secret	ss_X	32
X-Wing encapsulation key	pk	1216
X-Wing ciphertext	ct	1120
X-Wing shared secret	ss	32

The X-Wing encapsulation key is the byte concatenation $pk = pk_M \,\|\, pk_X$ (1184 + 32 = 1216 bytes). The X-Wing ciphertext is $ct = ct_M \,\|\, ct_X$ (1088 + 32 = 1120 bytes).

3.2 Combiner

The X-Wing combiner is computed verbatim per `draft-connolly-cfrg-xwing-kem`:

$$ss \;=\; \text{SHA3-256}(ss_M \,\|\, ss_X \,\|\, ct_X \,\|\, pk_X \,\|\, L_{\text{XW}}) \tag{1}$$

where the label L_{XW} is the 6-byte ASCII string `\./` concatenated with `/^\`:

$$L_{\text{XW}} \;=\; \texttt{0x5c 0x2e 0x2f 0x2f 0x5e 0x5c} \tag{2}$$

The label appears *last* in the SHA3-256 input. The input order $ss_M \| ss_X \| ct_X \| pk_X \| L_{\text{XW}}$ is normative and MUST NOT be permuted; the combiner is a fixed function of these five operands in this exact order.

3.3 Encapsulation

Given peer encapsulation key $pk = pk_M \,\|\, pk_X$, the sender:

1. Generates an ephemeral X25519 key pair (sk_X^e, pk_X^e) per RFC 7748, with the standard clamping of sk_X^e.
2. Computes $ct_X \leftarrow pk_X^e$ and $ss_X \leftarrow \text{X25519}(sk_X^e, pk_X)$. If ss_X is the all-zero 32-byte string the sender MUST abort with a key-exchange failure (low-order point rejection per RFC 7748 §6.1).
3. Calls ML-KEM-768 encapsulation against pk_M, producing (ct_M, ss_M) per FIPS 203.
4. Computes ss via Equation 1.
5. Outputs (ct, ss) where $ct = ct_M \,\|\, ct_X$.

3.4 Decapsulation

Given decapsulation material (sk_M, sk_X, pk_X) and ciphertext $ct = ct_M \,\|\, ct_X$, the receiver:

1. Splits ct into ct_M (first 1088 bytes) and ct_X (next 32 bytes). Any other length MUST be rejected.
2. Computes $ss_M \leftarrow \text{ML-KEM-768.Decap}(sk_M, ct_M)$ per FIPS 203. ML-KEM-768 implicit rejection is preserved: a malformed ct_M yields a pseudorandom ss_M and the protocol MUST proceed to the AEAD step, where the failure surfaces as an authentication-tag mismatch.
3. Computes $ss_X \leftarrow \text{X25519}(sk_X, ct_X)$. The all-zero output rejection of RFC 7748 §6.1 MUST be enforced.
4. Computes ss via Equation 1.

4 Identity (Normative)

4.1 Hybrid identity

A Z-Wing identity is the pair (I_E, I_M) where I_E is an Ed25519 key pair (RFC 8032; 32-byte public key, 32-byte seed, 64-byte signature) and I_M is an ML-DSA-65 key pair (FIPS 204; 1952-byte public key, 4032-byte private key, 3309-byte signature).

The 32-byte node identifier is

$$\text{NodeID} \;=\; \text{SHA3-256}(I_E.\text{pk} \,\|\, I_M.\text{pk})\,. \tag{3}$$

4.2 Canonical signing message

Both identity signatures cover the same canonical message:

$$M_\sigma \;=\; \texttt{"zwing/1 transcript "} \,\|\, T \tag{4}$$

where the literal prefix is the 19-byte ASCII string `"zwing/1 transcript "` (note the trailing space) and T is the 32-byte transcript hash defined in §5. The fixed prefix is a domain separator; signatures over M_σ are not valid for any other Lux protocol context.

4.3 Verification

A peer's identity is bound to the channel only if BOTH signatures verify against M_σ under the advertised public keys. Either signature failing aborts the handshake. Forging an accepted identity binding requires breaking BOTH Ed25519 (current ECC attacks, including quantum) AND ML-DSA-65 (lattice attacks).

5 Handshake (Normative)

5.1 Wire framing

Handshake messages are length-prefixed: a 4-byte big-endian unsigned length L followed by L bytes of payload. L MUST NOT exceed 2^{20} (1 048 576) bytes; receivers MUST close the underlying transport on any larger value.

5.2 HandshakeInit (initiator → responder)

Field	Type	Description	Bytes
`magic`	u32 BE	`0x5A57494E` (ASCII `"ZWIN"`)	4
`version`	u8	Protocol version, MUST be `0x01`	1
`flags`	u8	Reserved, MUST be `0x00`	1
`ed25519_pk`	bytes	Initiator's long-term Ed25519 public key	32
`mldsa_pk`	bytes	Initiator's long-term ML-DSA-65 public key	1952
`xwing_pk`	bytes	Initiator's ephemeral X-Wing encapsulation key	1216
`nonce_i`	bytes	Initiator handshake nonce (uniformly random)	32

Table 1: HandshakeInit. Fixed total length: 3239 bytes. No padding, no extensions.

5.3 HandshakeResponse (responder → initiator)

Field	Type	Description	Bytes
`magic`	u32 BE	`0x5A57494E`	4
`version`	u8	MUST equal initiator's version	1
`flags`	u8	Reserved, MUST be `0x00`	1
`ed25519_pk`	bytes	Responder's long-term Ed25519 public key	32
`mldsa_pk`	bytes	Responder's long-term ML-DSA-65 public key	1952
`xwing_ct`	bytes	X-Wing ciphertext encapsulated to initiator's key	1120
`nonce_r`	bytes	Responder handshake nonce	32
`ed25519_sig`	bytes	Responder's Ed25519 signature over M_σ	64
`mldsa_sig`	bytes	Responder's ML-DSA-65 signature over M_σ	3309

Table 2: HandshakeResponse. Fixed total length: 6515 bytes.

5.4 HandshakeFinish (initiator → responder, encrypted)

After deriving session keys, the initiator sends a single AEAD-protected frame (per §6) carrying:

Field	Type	Description	Bytes
`ed25519_sig`	bytes	Initiator's Ed25519 signature over M_σ	64
`mldsa_sig`	bytes	Initiator's ML-DSA-65 signature over M_σ	3309

Table 3: HandshakeFinish (plaintext fields). Encrypted as the first i→r data frame.

Remark 1. *Z-Wing is therefore one round trip on the wire: HandshakeInit and HandshakeResponse cross before any application data flows. The initiator's signature is delivered as the first AEAD frame, encrypted under the freshly derived session keys. This design preserves 1-RTT (application data MAY be appended to the same TCP segment as HandshakeFinish) while keeping the initiator identity hidden from passive observers.*

5.5 Transcript

The transcript hash T binds every byte either party put on the wire during the handshake:

$$T \;=\; \text{SHA3-256}(\texttt{HandshakeInit} \,\|\, \texttt{HandshakeResponseHeader}) \tag{5}$$

where `HandshakeResponseHeader` is the responder's message with `ed25519_sig` and `mldsa_sig` omitted (the first $4 + 1 + 1 + 32 + 1952 + 1120 + 32 = 3142$ bytes). Both peers compute T identically from their view of the wire. T is the value used in M_σ.

5.6 Key schedule

After decapsulation both peers hold the X-Wing shared secret ss (32 bytes). Session keys are derived via HKDF-SHA3-256 (RFC 5869 with SHA3-256 as the hash):

$$\text{PRK} \;=\; \text{HKDF-Extract}_{\text{SHA3-256}}(\,\text{salt} = T,\ \text{IKM} = ss\,) \tag{6}$$

$$K_{i\to r} \;=\; \text{HKDF-Expand}_{\text{SHA3-256}}(\text{PRK},\ \texttt{"zwing-i2r"},\ 32) \tag{7}$$

$$K_{r\to i} \;=\; \text{HKDF-Expand}_{\text{SHA3-256}}(\text{PRK},\ \texttt{"zwing-r2i"},\ 32) \tag{8}$$

The info labels are the 9-byte ASCII strings `"zwing-i2r"` (hex `7a 77 69 6e 67 2d 69 32 72`) and `"zwing-r2i"` (hex `7a 77 69 6e 67 2d 72 32 69`). Each derived key is 32 bytes, sized for ChaCha20.

5.7 State machines

Initiator	Responder
`INIT` → generate X-Wing keypair, nonce$_i$	`LISTEN`
Send HandshakeInit; transition to `WAIT_RESP`	Receive HandshakeInit; validate magic, version, flags, lengths
Receive HandshakeResponse; validate; decap ss; compute T, PRK, $K_{i\to r}$, $K_{r\to i}$	Encap ss; compute T, PRK, $K_{i\to r}$, $K_{r\to i}$; sign M_σ
Verify responder σ_E, σ_M over M_σ; abort on either failure	Send HandshakeResponse; transition to `WAIT_FINISH`
Sign M_σ; send HandshakeFinish (AEAD frame under $K_{i\to r}$, sequence 0); transition to `ESTABLISHED`	Receive first AEAD frame under $K_{i\to r}$, sequence 0; decrypt; verify initiator σ_E, σ_M over M_σ; transition to `ESTABLISHED`
Zero ephemeral X-Wing keys	Zero responder-side X-Wing decap material

Figure 2: Z-Wing handshake state machines.

Any validation failure — magic mismatch, version mismatch, non-zero `flags`, length-prefix overflow, low-order X25519 point, signature verification failure, AEAD tag failure — MUST close the underlying transport without a recovery message. There is no error frame.

5.8 Negotiation

There is none. Magic, version, KEM, identity scheme, KDF, AEAD, and transcript hash are fixed. A peer that does not understand `version=0x01` closes. Future versions allocate a new value.

6 Transport Frame (Normative)

After `ESTABLISHED`, every byte exchanged in either direction is carried in AEAD frames.

6.1 Frame format

Field	Type	Description	Bytes
`length`	u32 BE	Length of `ciphertext` including AEAD tag	4
`ciphertext`	bytes	ChaCha20-Poly1305 ciphertext ‖ 16-byte tag	L

Table 4: Z-Wing transport frame. Fixed 4-byte big-endian length prefix followed by AEAD output.

`length` MUST NOT exceed 2^{20} bytes. Receivers MUST close the transport on any larger value. The minimum `length` is 17 (one plaintext byte plus the 16-byte tag); zero-length plaintexts are not permitted.

6.2 Nonce derivation

Each direction maintains an independent 64-bit sequence counter $n_{i\to r}$, $n_{r\to i}$, both initialised to zero. The 12-byte ChaCha20-Poly1305 nonce for a frame is

$$\text{nonce} \;=\; \texttt{0x00 0x00 0x00 0x00} \,\|\, n^{(64)}_{\text{dir, BE}} \tag{9}$$

i.e. four zero bytes followed by the directional sequence counter as a big-endian unsigned 64-bit integer. The counter is incremented after each successful AEAD operation (encryption on the sender side, decryption on the receiver side).

6.3 Replay and ordering

Sequence counters are strictly increasing. Receivers MUST verify that the incoming counter equals the expected next value; any deviation MUST close the transport. Reordering, replay, gap, or duplication terminates the channel. Because the underlying transport is reliable and ordered (TCP, Unix, RNS link), this is an integrity property, not a packet filter.

6.4 Counter exhaustion

A direction that reaches $n_{\text{dir}} = 2^{64} - 1$ MUST close the transport before encrypting another frame. At a sustained 10^6 frames per second this exhausts in $\sim 5.85 \times 10^5$ years; the bound exists to make exhaustion a defined error rather than an undefined overflow.

6.5 Application MTU

Plaintext payload per frame MUST be at most $2^{20} - 16 = 1\,048\,560$ bytes (1 MiB minus the AEAD tag). `Read`/`Write` implementations MAY fragment larger application writes across multiple frames; the layering above Z-Wing (`net.Conn`) is byte-oriented, so frame boundaries are not visible to the application.

7 Security Analysis

7.1 Hybrid KEM security

The X-Wing combiner provides IND-CCA security if EITHER ML-KEM-768 OR the gap-CDH variant of X25519 (with the SHA3-256 random oracle assumption) remains hard. The construction and proof are `draft-connolly-cfrg-xwing-kem`; this LP adopts them by reference. An attacker MUST break both component primitives to recover the session shared secret ss.

7.2 Forward secrecy

The X-Wing key pair is ephemeral per handshake and zeroed once session keys are derived. The 32-byte ss is consumed by HKDF-Extract and then discarded. Compromise of long-term Ed25519 or ML-DSA-65 identity keys does not retroactively decrypt prior sessions because neither participates in ss derivation; identities only sign M_σ.

7.3 Identity binding

M_σ contains the transcript hash T, which commits to every byte of HandshakeInit and the unsigned portion of HandshakeResponse, including the X-Wing public key, the ciphertext, both nonces, and both peers' long-term identities. A signature over M_σ is therefore not replayable in any other session. Identity forgery requires breaking BOTH Ed25519 and ML-DSA-65 simultaneously; this is the defence-in-depth property the parallel-signature construction provides.

7.4 Downgrade resistance

There is no negotiation surface. Magic, version, KEM, AEAD, transcript hash, and identity scheme are fixed bytes on the wire. An active attacker cannot induce either peer to use a weaker construction because there is no weaker construction in the protocol grammar.

7.5 Quantum threat model

- **Store-now-decrypt-later.** Mitigated by ML-KEM-768 contributing to *ss*. A future cryptanalytically relevant quantum computer (CRQC) capable of breaking X25519 cannot recover *ss* without also breaking ML-KEM-768.
- **Real-time impersonation under CRQC.** Mitigated by ML-DSA-65. Without ML-DSA forgery the attacker cannot produce an accepted handshake even if Ed25519 is broken in real time.
- **Long-term identity leak.** Outside the channel's threat model; addressed at the layer that issues identities (Lux KMS / I-Chain).

7.6 Side channels

ML-KEM-768 and ML-DSA-65 implementations MUST be constant-time per FIPS 203 §3.3 and FIPS 204 §3.5 respectively. ChaCha20-Poly1305 is the chosen AEAD specifically because table-free constant-time implementations are routine. AES-GCM is not part of this LP.

8 Layering: ZAP on Z-Wing

The ZAP wire format (`luxfi/api/zap`) requires only a `net.Conn`; `zap.NewConn` accepts any value satisfying that interface. Z-Wing implements `net.Conn`. ZAP-over-Z-Wing is therefore the literal composition of two independent specifications; this LP does not modify ZAP and ZAP does not constrain Z-Wing.

Deployment	Composition
TCP only	`zap.NewConn(net.Dial("tcp", addr))` — no PQ, no PFS, no auth.
Z-Wing	`zap.NewConn(zwing.Dial(ctx, addr, ident, peerPK))` — hybrid PQ, PFS, mutual auth.
Z-Wing on RNS	`zap.NewConn(zwing.NewChannel(rns.Dial(...)))` — as above plus mesh routing.

The same ZAP RPC surface (opcodes, request-ID demultiplexing, message framing) operates identically in all three deployments. Operators choose the channel; applications do not.

9 Migration from LP-9701

LP-9702 supersedes LP-9701. The migration is a hard cut: there is no negotiation between LP-9701 and LP-9702 wire formats, and no backward-compatible mode in the Z-Wing handshake.

- **Identity reuse.** Existing Ed25519 and ML-DSA-65 identity keys carry forward unchanged. The Z-Wing `NodeID` is recomputed because it now binds both keys explicitly; deployments MUST treat the new `NodeID` as the canonical identifier and update peer lists.
- **Wire format.** The LP-9701 `MsgTypeLinkRequest` (`0x01`) byte is incompatible with Z-Wing's magic `0x5A57494E`. A peer receiving the wrong opening bytes MUST close.

- **Combiner.** The LP-9701 combiner (HKDF-SHA256 over X25519 ECDH and a sorted-public-key hash) is replaced by the IETF X-Wing combiner (Equation 1). No LP-9701 session key is interconvertible with a Z-Wing session key.
- **RNS integration.** Z-Wing-on-RNS is specified by a separate package (`luxfi/zwing/rns`) and is out of scope for this LP. The LP-9701 RNS announce/discovery layer continues to operate; only the link-layer handshake changes.
- **Deprecation.** LP-9701 is deprecated by this document. Implementations MUST NOT initiate new LP-9701 sessions once LP-9702 is deployed. Coordinated rollout of all validators in a network is the canonical migration path.

10 IANA / Wire Considerations

- **TCP port.** 9999, the canonical Lux ZAP port. Z-Wing completes its handshake before any ZAP traffic; the same port carries both phases of the connection.
- **ALPN.** Where ALPN negotiation is present (e.g. inside a TLS/QUIC stream that Z-Wing rides above), the identifier is `"zwing/1"`. This LP does not require ALPN.
- **Magic.** `0x5A57494E` (`"ZWIN"`). This prefix MAY be used by transport-layer demultiplexers to distinguish Z-Wing from other byte-stream protocols on the same port.

11 References

References

[1] National Institute of Standards and Technology. *FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard (ML-KEM)*. https://doi.org/10.6028/NIST.FIPS.203.

[2] National Institute of Standards and Technology. *FIPS 204: Module-Lattice-Based Digital Signature Standard (ML-DSA)*. https://doi.org/10.6028/NIST.FIPS.204.

[3] National Institute of Standards and Technology. *FIPS 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. https://doi.org/10.6028/NIST.FIPS.202.

[4] A. Langley, M. Hamburg, S. Turner. *RFC 7748: Elliptic Curves for Security*. IETF, January 2016. https://www.rfc-editor.org/rfc/rfc7748.

[5] S. Josefsson, I. Liusvaara. *RFC 8032: Edwards-Curve Digital Signature Algorithm (EdDSA)*. IETF, January 2017. https://www.rfc-editor.org/rfc/rfc8032.

[6] Y. Nir, A. Langley. *RFC 8439: ChaCha20 and Poly1305 for IETF Protocols*. IETF, June 2018. https://www.rfc-editor.org/rfc/rfc8439.

[7] H. Krawczyk, P. Eronen. *RFC 5869: HMAC-based Extract-and-Expand Key Derivation Function (HKDF)*. IETF, May 2010. https://www.rfc-editor.org/rfc/rfc5869.

[8] D. Connolly, P. Schwabe, B. Westerbaan. *X-Wing: General-Purpose Hybrid Post-Quantum KEM*. `draft-connolly-cfrg-xwing-kem`, IRTF CFRG. https://datatracker.ietf.org/doc/draft-connolly-cfrg-xwing-kem/.

[9] Lux Industries. *LP-9701: Reticulum Network Stack with Hybrid Post-Quantum Link*. Superseded by this document.