

EchoPulse PQ Forward Secrecy – TLS Handshake Extension and Enforcement (v3.0)

1. Overview

This document specifies the integration of EchoPulse's Post-Quantum Forward Secrecy (PQ-FS) into the TLS 1.3 handshake protocol. Building upon the foundational structure defined in "EchoPulse PQ Forward Secrecy – Structural Foundation (v3.0)" (File 43) and the "EchoPulse PQ Forward Secrecy – Key Derivation Model (v3.0)" (File 44), this specification details the necessary TLS handshake extensions and negotiation mechanisms. The core innovation lies in the deterministic derivation and enforcement of a session-specific, strictly monotonic `forward_entropy_epoch`, which directly binds each KEM operation to a unique Graph-Time State (Gt), thereby providing PQ-FS.

2. Protocol Architect: Protocol Modifications

To support EchoPulse PQ-FS, the TLS 1.3 handshake utilizes existing extension points while formalizing the derivation and usage of the `forward_entropy_epoch` as a session-bound entropy index.

2.1. Negotiation Process using `echopulse_parameters`

The negotiation of EchoPulse PQ-FS is seamlessly integrated into the existing `echopulse_parameters` extension, defined within TLS 1.3's KeyShare mechanism.

- **ClientHello Logic:**

- If the client supports EchoPulse, it includes a KeyShareEntry for the DRAFT_ECHOPULSE_KEM NamedGroup (or similar EchoPulse-specific NamedGroup identifier).
- Within this KeyShareEntry, the client *MUST* include the `echopulse_parameters` extension.
- The `echopulse_parameters` extension *SHOULD* explicitly signal the client's support for PQ-FS by including a dedicated flag or a specific `mutation_schedule_id` value known to imply PQ-FS capability. For instance, a new EchoPulseFeatures bitmask could be introduced within `echopulse_parameters`, with a bit for PQFS_SUPPORTED.
- The `key_exchange` field of the KeyShareEntry will contain PKEPt, derived using a Gt based on an *initial* `forward_entropy_epoch` (tpre-SH) computed from available ClientHello elements.

- **ServerHello Logic:**

- If the server selects the DRAFT_ECHOPULSE_KEM NamedGroup, it *MUST* include an echopulse_parameters extension in its ServerHello (if present in ClientHello) mirroring the selected parameters.
- The ServerHello *MUST* include a KeyShareEntry containing CTEPt, which is encapsulated using a Gt derived from the *final* forward_entropy_epoch (tpost-SH) as confirmed by the ServerHello contents.
- If the client signaled PQ-FS support and the server also supports and selects it, the server *MUST* confirm this selection within its echopulse_parameters (e.g., by echoing the PQFS_SUPPORTED flag).

2.2. Formalizing forward_entropy_epoch

The forward_entropy_epoch is a **derived, implicit field**, meaning it is not explicitly transmitted over the wire as a separate parameter. Instead, both client and server independently compute it from shared, unambiguous handshake data. It directly corresponds to the time index t in Gt.

- **Derivation:** The exact derivation of forward_entropy_epoch (t) is crucial for synchronization and non-malleability. It relies on a cryptographically secure hash of specific, ordered handshake messages and agreed-upon random nonces. (Detailed in Section 3).
- **Usage:** Once computed, t is used by both parties to:
 1. Deterministically generate the Graph-Time State $G_t = \mu(G_0, t)$.
 2. Perform all EchoPulse KEM operations (encapsulation and decapsulation) using this specific Gt. This ensures that the key derivation KEPt is uniquely tied to the session's temporal context.
- **Binding to Gt and PQ-FS Logic:** The binding is fundamental: Gt is the graph state at epoch t . Any operation (path traversal r , HORM interaction, venc derivation) is performed on this specific Gt. This ensures that even if an adversary obtains long-term keys, the unique Gt for any past session cannot be re-derived or simulated without the exact session context, thus enforcing PQ-FS.

2.3. Implicit vs. Explicit Elements

- **Explicitly Negotiated:**
 - NamedGroup (DRAFT_ECHOPULSE_KEM) in ClientHello and ServerHello.
 - echopulse_parameters extension (containing graph_id, mutation_schedule_id, symbol_path_length, hash_mode, and potentially EchoPulseFeatures flags like PQFS_SUPPORTED).
 - KeyShareEntry fields: key_exchange (containing PKEPt or CTEPt).

- **Implicitly Derived:**

- forward_entropy_epoch (t): Derived from the handshake transcript.
- Graph-Time State (Gt): Deterministically generated by applying $\mu(GO, t)$.
- Session Shared Secret (KEPt): Derived using the KEM model on Gt.

2.4. Updated ClientHello and ServerHello Logic with PQ-FS Context

Code-Snippet

sequenceDiagram

participant C as Client

participant S as Server

C->>S: ClientHello (NamedGroup: DRAFT_ECHOPULSE_KEM,
echopulse_parameters {PQFS_SUPPORTED}, KeyShare: PK_EP_t_pre_SH)
note right of C: Calculates t_pre_SH from ClientHello.random and other fields.
note right of C: Generates PK_EP_t_pre_SH on G_t_pre_SH.

S->>C: ServerHello (NamedGroup: DRAFT_ECHOPULSE_KEM,
echopulse_parameters {PQFS_SUPPORTED}, KeyShare: CT_EP_t_post_SH)
note left of S: Calculates t_post_SH from ClientHello + ServerHello.random + other fields.
note left of S: Encapsulates CT_EP_t_post_SH on G_t_post_SH, deriving K_EP_t_post_SH.

C->>S: [EncryptedExtensions, Certificate, CertificateVerify, Finished]
note right of C: Calculates t_post_SH (same as Server).
note right of C: Decapsulates CT_EP_t_post_SH on G_t_post_SH, deriving K_EP_t_post_SH.
note right of C: If K_EP_t_post_SH derivation fails (e.g., v_dec mismatch), ABORT.

S->>C: [Certificate, CertificateVerify, Finished]
note left of S: Uses K_EP_t_post_SH to decrypt Finished.

2.5. Fallback Behavior

- **No PQ-FS Support:** If either client or server does not support the

PQFS_SUPPORTED flag (or equivalent signaling), but both support a basic EchoPulse NamedGroup, the handshake *MAY* proceed with the standard EchoPulse KEM as defined in v1.0/v2.x, without PQ-FS guarantees. This is an explicit negotiation failure for PQ-FS, not a protocol error.

- **NamedGroup Mismatch:** If the client proposes DRAFT_ECHOPULSE_KEM but the server does not select it (or vice-versa), the handshake falls back to other negotiated NamedGroups or aborts, as per standard TLS 1.3 behavior.

2.6. Parameter Compatibility with TLS NamedGroup and KeyShareEntry

The EchoPulse PQ-FS integration leverages the existing NamedGroup (e.g., DRAFT_ECHOPULSE_KEM) and KeyShareEntry structures. The specific EchoPulse parameters (e.g., graph_id, mutation_schedule_id, symbol_path_length, hash_mode) are conveyed within the echopulse_parameters extension, ensuring full compatibility with TLS 1.3's extensibility model.

3. Cryptographic Negotiation Strategist: Entropy Epoch Derivation

The secure and deterministic derivation of the forward_entropy_epoch (time index t) by both client and server is paramount for PQ-FS. It must occur without additional round trips and be resilient against manipulation.

3.1. $t = \text{derive_entropy_epoch}(\dots)$ Function

The time index t is derived from a cryptographically strong hash of specific, ordered, and non-repeating handshake elements. This ensures that t is unique for each session, synchronized between parties, and resistant to tampering.

$t = \text{KDF-Extract}(\text{SALT}, \text{IKM})$

Where:

- **IKM (Input Keying Material):** A concatenation of high-entropy, session-unique elements:
 - ClientHello.random
 - ServerHello.random
 - Transcript-Hash(ClientHello...ServerHello) (the cumulative hash of all handshake messages exchanged up to the point of t derivation). This ensures that t is bound to the entire handshake context.
 - Optionally, a session-specific ephemeral nonce (e.g., derived from the PreSharedKey binder if PSK is used, or a session counter).
- **SALT:** A fixed, publicly known value (e.g., EchoPulse_PQFS_SALT) or a value

derived from the `graph_id` to further diversify `t`.

- **KDF-Extract:** A Cryptographically Secure Key Derivation Function (KDF) in extract-only mode (e.g., HKDF-Extract using the `hash_mode` negotiated).

Example Derivation Point: The derivation of `t` can be anchored after the `ServerHello` message. At this point, `ClientHello.random`, `ServerHello.random`, and the Transcript-Hash up to `ServerHello` are all available to both parties.

3.2. Security Rationale Against Manipulation or Ambiguity

- **Non-Malleability:** By including Transcript-Hash as a core component of IKM, any alteration to previous handshake messages by an on-path adversary would result in a different Transcript-Hash, leading to a different `t` for the adversary. This mismatch would cause the KEM decapsulation to fail (as `Gt` would differ) and the subsequent Finished message verification to fail.
- **Unpredictability:** The inclusion of `ClientHello.random` and `ServerHello.random` ensures that `t` is highly unpredictable for future sessions. Even if an adversary can predict one party's random value, they cannot predict the other's, preventing them from pre-computing `Gt` for future sessions.
- **No Extra Round Trips:** The derivation process uses existing handshake elements, requiring no additional messages or round trips, maintaining TLS 1.3's 1-RTT or 0-RTT characteristics.
- **Deterministic:** Using a standard, deterministic KDF ensures that both client and server will always derive the exact same `t` given the same inputs.

4. Negotiation Flow

The negotiation flow within TLS 1.3 for EchoPulse PQ-FS relies on the capabilities signaled via the `echopulse_parameters` extension and the shared, deterministic computation of the `forward_entropy_epoch`.

1. **Client Hello:** Client proposes `DRAFT_ECHOPULSE_KEM` with `echopulse_parameters` indicating `PQFS_SUPPORTED`. Client calculates an initial `tpre-SH` and generates `PKEPtpre-SH` on `Gtpre-SH`. This `PKEPtpre-SH` is largely symbolic; the *actual* KEM operation will use `Gtpost-SH`.
2. **Server Hello:** Server selects `DRAFT_ECHOPULSE_KEM` and responds with `echopulse_parameters` mirroring `PQFS_SUPPORTED`. Both client and server *simultaneously* compute the final `forward_entropy_epoch` (`tpost-SH`) based on `ClientHello.random`, `ServerHello.random`, and Transcript-Hash. Server then generates `Gtpost-SH` and encapsulates `CTEPtpost-SH` for client's `PKEPtpre-SH`.

Server immediately derives KE_{TP}post-SH.

3. **Client Processing:** Upon receiving ServerHello, client computes t_{post}-SH using the same logic as the server, derives G_tpost-SH, and then decapsulates CTE_{TP}post-SH using its secret key on G_tpost-SH to derive KE_{TP}post-SH.
4. **Key Schedule & Finished:** Both parties proceed with the TLS 1.3 Key Schedule using the derived KE_{TP}post-SH as the primary shared secret for deriving traffic keys and verifying Finished messages.

This two-stage t derivation (initial and final) allows for the KeyShareEntry to be sent in the ClientHello while ensuring the *actual* session key is bound to the full ServerHello context for maximum entropy and forward secrecy.

5. Implementation Security Lead: Validation and Downgrade Protection

Robust implementation of EchoPulse PQ-FS requires strict validation and explicit protection against various attacks.

5.1. No Weakening of Session Confidentiality

The PQ-FS parameters and derivation logic (t , G_t , $\mu(G, t)$) are designed to *enhance* confidentiality by providing post-quantum forward secrecy. They do not introduce any known vulnerabilities that would weaken session confidentiality. The core KEM operation remains robust and resistant to classical and quantum attacks (assuming the underlying assumptions of EchoPulse KEM hold). Any intermediate values (v_{enc} , v_{dec} , h_{local}) are used within the secure key derivation function and are not meant to leak information. Their leakage resilience is maintained through constant-time implementations and their dependence on the secret path r and dynamic G_t .

5.2. Replay Detection at Handshake Level

While EchoPulse's KEM provides inherent replay resistance at the KEM layer (as detailed in "EchoPulse PQ Forward Secrecy – Key Derivation Model (v3.0)" and through the "EchoPulse Replay Verifier CLI"), the handshake level also implicitly aids this:

- **forward_entropy_epoch Mismatch:** If an adversary attempts to replay a captured ClientHello and ServerHello (including their KeyShare data) in a new session, the derived forward_entropy_epoch (t) will be different. This is because t depends on ClientHello.random, ServerHello.random, and Transcript-Hash, all of which will change with a new session.
- **KEM Decapsulation Failure:** A mismatch in t leads to a mismatch in G_t .

Consequently, the KEM decapsulation of the replayed CTEPt on the new Gt' will fail (i.e., $v_{dec} \neq v_{enc_id}$), causing the handshake to abort.

5.3. Resistance to Downgrade Attacks (e.g., Stripping Entropy)

- **PQFS_SUPPORTED Flag:** The explicit signaling via the PQFS_SUPPORTED flag within echopulse_parameters is crucial. An adversary attempting to strip this flag (a typical downgrade attack) would result in the server either not selecting DRAFT_ECHOPULSE_KEM or proceeding without PQ-FS, which the client might detect as a policy violation.
- **Transcript-Hash Integrity:** The reliance of t on the Transcript-Hash makes it resistant to manipulation. An adversary attempting to alter handshake messages (e.g., to strip random values or alter mutation_schedule_id to influence t) would cause the Transcript-Hash to mismatch, leading to an t derivation mismatch and handshake failure.
- **Must/Should Implementations:**
 - **MUST:** Both client and server *MUST* compute the forward_entropy_epoch (t) deterministically using the specified KDF and handshake elements. Any mismatch in the computed t *MUST* lead to a fatal alert and handshake termination.
 - **MUST:** The KEM decapsulation process *MUST* validate v_{dec} against v_{enc_id} . A mismatch *MUST* lead to a fatal alert.
 - **SHOULD:** Clients and servers *SHOULD* implement policies to prefer PQ-FS-enabled EchoPulse handshakes. If a PQ-FS capable peer attempts a non-PQ-FS EchoPulse handshake (after successful negotiation of PQ-FS capability), it *SHOULD* trigger an alert (e.g., illegal_parameter or a custom alert).

6. Meta Role – Compatibility and Integration

This document meticulously integrates EchoPulse PQ-FS into the TLS 1.3 framework, ensuring consistency with prior EchoPulse specifications and maintaining TLS 1.3 compliance.

6.1. Compatibility with Prior Files

- **43EchoPulse.PQForwardSecrecy.Structure:** This document directly implements the structural foundation by defining how the forward_entropy_epoch (the monotonic index t) is derived within the TLS handshake and how it binds to the Graph-Time State (Gt).
- **44EchoPulse.PQFS.KeyDerivation.Model:** The negotiation and enforcement

specified here ensure that the inputs required for the detailed key derivation logic (e.g., G_t , proper Transcript-Hash for seed_H) are securely established and available to both parties. The v_{enc} validation at the KEM layer is triggered by the outcome of this handshake.

6.2. Reuse of Core Terms

All core EchoPulse terms such as G_t , $\mu(G,t)$, r , KEPt , v_{enc} , v_{dec} , HORM , and the newly formalized $\text{forward_entropy_epoch}$ (as the precise derivation of t) are used consistently and precisely throughout this document, reinforcing the cohesive nature of the EchoPulse framework.

6.3. Modular Formatting and Semantic Clarity

The document adheres to a modular structure with clear section headers, precise technical language, and the use of sequence diagrams to illustrate the handshake flow, ensuring high semantic clarity and readiness for integration into the master dossier.

6.4. Integration without Breaking TLS 1.3 Compliance

The proposed extensions and modifications to TLS 1.3 are designed to be fully compliant with the existing TLS 1.3 specification. They leverage the KeyShare and extension mechanisms as defined, without introducing new message types or altering fundamental TLS 1.3 state machines. Fallback mechanisms ensure interoperability with non-PQ-FS capable peers. This approach allows for a graceful transition and incremental adoption of EchoPulse's PQ-FS capabilities within the existing TLS ecosystem.