

EchoPulse PQ Forward Secrecy – Key Derivation Model (v3.0)

1. Introduction

The previous iteration of EchoPulse (v3.0, "EchoPulse PQ Forward Secrecy – Structural Foundation") established the foundational concept of Post-Quantum Forward Secrecy (PQ-FS) through time-bound symbolic graph mutation. This document, "EchoPulse PQ Forward Secrecy – Key Derivation Model (v3.0)," delves into the precise cryptographic mechanics of session-specific key derivation within this architecture. It formally defines the encapsulation and decapsulation processes, detailing how the session-specific shared secret KEPT is securely derived from the unique Graph-Time State Gt. Emphasis is placed on the interaction with the Hash Oracle Replacement Model (HORM), the role of path-bound randomness, and the optimization for constrained embedded environments. Furthermore, this document audits the derivation model's resilience against various threats, ensuring robust post-quantum security guarantees.

2. Cryptographic Systems Designer: Key Derivation Structure

The core of EchoPulse's PQ-FS is the deterministic derivation of a unique session key KEPT for each communication session, inextricably linked to a specific graph-time state Gt. This section formalizes the encapsulation and decapsulation processes.

2.1. Input/Output Summary of the Derivation Function

The central function for key derivation in EchoPulse is essentially a KEM operation that takes a public key, private key (for decapsulation), ciphertext (for decapsulation), and crucially, the specific graph-time state Gt as context.

- **Encapsulation Input:**
 - PKEPT: EchoPulse public key for session t. This inherently includes parameters defining Gt (or implies its derivation).
 - Gt: The Graph-Time Context at time t, derived from the monotonic entropy index t.
 - r: A randomly generated symbolic path (sequence of symbols), length `symbol_path_length`.
- **Encapsulation Output:**
 - CTEPT: EchoPulse ciphertext for session t.
 - KEPT: Derived shared secret for session t.
- **Decapsulation Input:**
 - SKEPT: EchoPulse private key for session t. This implicitly allows derivation of

the corresponding PKEPt.

- CTEPt: EchoPulse ciphertext received for session t.
- Gt: The Graph-Time Context at time t, derived deterministically from the handshake.

- **Decapsulation Output:**

- KEPt: Recovered shared secret for session t.

2.2. Formal Derivation Logic

The derivation of KEPt relies on the deterministic traversal of the symbolic graph Gt guided by the symbolic path r, and the interaction with the Hash Oracle Replacement Model (HORM).

A. Graph-Time State Derivation:

Both parties deterministically compute Gt at the beginning of each session.

$G_t = \mu(G_0, t)$

where G0 is the initial base graph (graph_id), and t is the strictly monotonic entropy index derived from handshake elements (e.g., Transcript-Hash, session counters) as discussed in "EchoPulse PQ Forward Secrecy – Structural Foundation (v3.0)". The mutation function $\mu(G, t)$ ensures that each Gt is unique and irrecoverable from Gt' for $t \neq t'$.

B. Encapsulation (Sender's Side):

1. **Generate Ephemeral Private Key:** The encapsulator (typically the client in TLS 1.3's KEM model, or the server when receiving a Client's PK) generates a session-specific ephemeral secret key SKEPt. This key is used internally to generate the random symbolic path r and potentially to seed the HORM.
 - $SKEPt \leftarrow \text{Random}(\text{SecurityParameter})$
2. **Generate Symbolic Path:** A random symbolic path $r = (s_1, s_2, \dots, s_L)$ of length $L = \lfloor \text{text}\{\text{symbol_path_length}\} \rfloor$ is generated, typically derived from SKEPt or a fresh random source. This path guides the graph traversal.
3. **Graph Traversal and Encapsulation:**
 - Start at an initial encapsulation vertex $v_{enc_start} \in V(G_t)$.
 - For each symbol s_i in r:
 - The HORM processes the current state (v_{i-1}, s_i) on Gt to produce a new vertex v_i and a local hash output $h_{local, i}$.
 - $(v_i, h_{local, i}) = \text{HORM}(G_t, v_{i-1}, s_i, \text{seedH})$
 - Here, seedH is a session-specific seed derived from the handshake transcript to ensure per-session unpredictability of HORM outputs.
 - The final vertex reached after traversing r is $v_{enc} = v_L$.
4. **Ciphertext Formation:** The ciphertext CTEPt consists of:

- `venc_id`: Identifier of the final encapsulation vertex `venc`.
 - `symbolic_path_sequence`: The actual symbolic path `r`.
 - Optionally, `public_graph_params`: Parameters defining `Gt`, or implicitly derived (e.g., `graph_id`, `mutation_schedule_id`).
5. **Shared Secret Derivation:** The shared secret `KEPt` is derived by hashing a combination of the ephemeral KEM secret, the public key, and the path's contribution to the graph state.
- $K_{EP_t} = \text{KDF}(\text{shared_secret_material} || \text{context_info})$
 - The `shared_secret_material` here includes the HORM's derived outputs and elements tied to `SKEPt`. A common approach is: $K_{EP_t} = \text{KDF}(\text{HORM_Output_Aggregate} || PK_{EP_t} || CT_{EP_t})$ where `HORM_Output_Aggregate` is a combination of relevant `hlocal,i` values or a final state-dependent hash derived from the traversal using `Gt`.

C. Decapsulation (Receiver's Side):

1. **Extract Ciphertext Components:** The decapsulator (typically the server in TLS 1.3) receives `CTEPt` and extracts `venc_id` and `symbolic_path_sequence` (`r`).
2. **Graph-Time State Derivation:** The decapsulator *independently* computes the identical `Gt` using the same strictly monotonic entropy index `t` derived from the handshake transcript.
 - $G_t = \mu(G_0, t)$
3. **Graph Traversal and Decapsulation:**
 - Start at the same initial vertex $v_{dec_start} \in V(G_t)$.
 - For each symbol `si` in the received `r`:
 - The HORM processes (v_{i-1}, s_i) on the *same* `Gt` to produce v_i' and `hlocal,i'`.
 - $(v_i', hlocal, i') = \text{HORM}(G_t, v_{i-1}, s_i, \text{seedH})$
 - The final vertex reached is $v_{dec} = v_{L'}$.
4. **Validation and Shared Secret Recovery:**
 - **Validation:** The decapsulator checks if `vdec` matches the `venc_id` provided in `CTEPt`. If they do not match, decapsulation fails, indicating a corrupted ciphertext or an attempt to replay an old ciphertext on a new `Gt`. This is the primary replay detection mechanism.
 - **Shared Secret Derivation:** If validation succeeds, the decapsulator derives `KEPt` using the same KDF and aggregated HORM outputs as the encapsulator. $K_{EP_t} = \text{KDF}(\text{HORM_Output_Aggregate}' || PK_{EP_t} || CT_{EP_t})$ Since `Gt` is identical and the HORM is deterministic, $\text{HORM_Output_Aggregate}'$ will be identical to $\text{HORM_Output_Aggregate}$.

text{HORM_Output_Aggregate}\$, leading to successful recovery of KE_{Pt}.

2.3. Path-Bound Randomness Contribution

The randomness of the derived shared secret KE_{Pt} is fundamentally bound to the unpredictability of the symbolic path r and its interaction with the dynamically evolving graph G_t through the HORM.

- **Symbolic Path r :** The initial random generation of r provides the primary source of entropy for the session key. Its length L (symbol_path_length) directly influences the security level.
- **Dynamic Graph G_t :** Even if an adversary could somehow predict parts of r , the fact that r is processed on a unique and evolving G_t (derived from a strictly monotonic entropy index t) makes it computationally infeasible to precompute or reverse the path-to-key mapping. The HORM's behavior is dependent on the specific G_t .
- **HORM Output:** The h_{local} outputs from the HORM, which contribute to the final KDF input, are also session-specific due to seed_H and G_t . This adds further entropy and makes it harder for an adversary to link or predict KE_{Pt} across sessions.

2.4. Interaction with Hash Oracle Replacement Model (HORM)

The HORM is central to EchoPulse's key derivation. As refined in EchoPulse v2.1 Audit Fix, the HORM takes (G_t, v, s, seed_H) as inputs and outputs (v', h_{local}).

- **Deterministic Mutation (G_t):** The HORM operates on the dynamically mutated G_t , which is unpredictable for future sessions. This provides the time-bound element of PQ-FS.
- **Symbolic Path Traversal:** The HORM is responsible for executing the symbolic path r on G_t , mapping input symbols to graph transitions and state changes.
- **Local Hash Output (h_{local}):** The h_{local} output from the HORM for each step of the traversal contributes to the shared secret derivation. This output combines the structural properties of G_t with the symbolic input s and a session-specific seed, ensuring that even if one step of traversal is known, the cumulative effect leading to KE_{Pt} remains secure.

3. Embedded Systems Protocol Engineer: Embedded Execution Constraints

EchoPulse's key derivation model is designed with extreme resource constraints in mind, targeting devices with ≤ 9 KB RAM and ≤ 15 KB ROM. This is achieved through

optimized design principles.

3.1. Deterministic Memory Access Patterns and Minimal Intermediate State Usage

- **Graph Representation:** Gt is typically represented in a highly compact, perhaps adjacency-list or compressed matrix format, optimized for direct lookup. The mutation function $\mu(G,t)$ itself is designed to apply changes incrementally and deterministically, minimizing the need for full graph re-computation.
- **In-Place Mutation:** Graph mutations are performed either in-place or with minimal intermediate buffering. This avoids dynamic memory allocation and reduces peak RAM usage. The "Mutation Buffer Compression" (as discussed in v2.1 Audit Fix) implies efficient handling of graph updates, further reducing memory footprint.
- **Symbolic Path Processing:** The symbolic path r is processed symbol-by-symbol, avoiding the need to store the entire sequence of intermediate v_i states in RAM. Only the current v_{i-1} and the next v_i are needed for each step.
- **Hash State Management:** The HORM's internal hash function state (e.g., for SHA256 or BLAKE2s) is typically fixed-size and processed iteratively, again preventing large memory requirements.

3.2. Linear or Sublinear Overhead for Graph Traversal and Mutation

- **Graph Traversal:** Processing a symbolic path r of length L involves L steps of HORM operations. Each HORM operation involves a deterministic lookup/computation within Gt. If Gt is well-indexed (e.g., using hash tables or direct addressing for nodes/edges), each step can be performed in $O(1)$ or $O(\log N)$ time, where N is the number of nodes. Thus, total traversal is $O(L)$ or $O(L \log N)$.
- **Graph Mutation:** The mutation function $\mu(G,t)$ is designed to be efficient. It applies a predefined `mutation_schedule_id` to G_0 for t steps. Each step of mutation is computationally light, involving specific, pre-determined alterations to graph structure. For a small t , the mutation overhead is minimal, often $O(1)$ or $O(\text{size_of_mutation_patch})$ per step. This ensures that dynamic graph evolution remains feasible on constrained devices. The "Mutation Cost Heatmap" (from v2.0, "EchoPulse Performance Heatmap") is designed to monitor this specific cost, ensuring it remains low and consistent.

3.3. Constant-Time Implementation and State-Bound Buffer Sizing

- **Constant-Time Operations:** All cryptographic operations, particularly the HORM and the KDF, must be implemented in constant time to prevent side-channel attacks (SCA). This means that the execution time should not depend on secret values (like SKEPt or the specific symbols in r). EchoPulse's deterministic graph traversal and pre-defined mutation schedules facilitate this. The "EchoPulse Performance Heatmap" aims to identify any timing anomalies indicative of non-constant-time behavior.
- **State-Bound Buffer Sizing:** Memory buffers are pre-allocated to fixed sizes based on the maximum expected graph dimensions and path length. This avoids dynamic memory allocation, which can be unpredictable and inefficient in embedded systems. For example, `symbol_path_length` is a negotiated parameter, allowing for pre-allocation. Intermediate values like `v_enc_id`, `symbolic_path_sequence`, and the output of KDF are fixed-size buffers, designed to fit within the stringent RAM constraints.

4. Threat Model & Resilience Auditor: Security Resilience

The EchoPulse key derivation model, with its PQ-FS structure, offers strong resilience against a range of cryptographic threats.

4.1. Resistance to Key Compromise (Long-Term + Ephemeral)

- **Long-Term Key Compromise:** Even if an adversary compromises a party's long-term signing key or static EchoPulse private key (SKLT) in the future, this does not compromise past session keys (KEPt). This is the essence of PQ-FS. The derivation of KEPt relies on the specific, time-bound G_t , which is unique to each session and cannot be re-derived or reverse-engineered from the compromised SKLT or past knowledge of G_0 . The `forward_entropy_epoch` effectively "burns" the graph state after each session.
- **Ephemeral Key Compromise:** If a session's ephemeral secret key (SKEPt) is compromised, only that specific session's KEPt is at risk. Since SKEPt is ephemeral and distinct for each session, its compromise does not reveal the long-term key, nor does it allow an adversary to compute KEPt' for any other session t' .

4.2. Resistance to Replay and Key Prediction Across Sessions

- **Replay Resistance:** This is an inherent property of EchoPulse's design. A ciphertext CTEPt from session t is decapsulatable *only* on G_t . If an adversary attempts to replay CTEPt in a later session t' , the decapsulation process will

occur on $G_{t'}$ where $G_{t'o}=G_t$. This will result in a different v_dec (or failure to find a valid path), causing the validation check ($vdec==venc_id$) to fail. The "EchoPulse Replay Verifier CLI" is a tool explicitly designed to test and confirm this resistance.

- **Key Prediction:** Due to the strictly monotonic and unpredictable nature of the entropy index t , and thus G_t , it is computationally infeasible for an adversary to predict future graph states or future session keys $KEPt'$ even if they have knowledge of past G_t states and $KEPt$ values.

4.3. Non-Linkability of Session Keys ($KEPt_o=f(KEPt-1)$)

The design actively ensures that session keys are non-linkable:

- The primary source of randomness, the symbolic path r , is freshly generated for each session.
- The underlying graph G_t changes with each session, making the input-output mapping of the HORM (and thus the KDF inputs) unique to each t .
- The seed $_H$ for the HORM is session-specific, further diversifying intermediate values. This ensures that observing $KEPt$ provides no information about $KEPt-1$ or any other $KEPt'$, establishing strong unlinkability and preventing the derivation of a chain of keys.

4.4. Leakage-Resilience of Intermediate Values

EchoPulse prioritizes leakage resilience for intermediate values like v_enc , v_dec , and h_local :

- **Deterministic vs. Secret:** While v_enc and v_dec are derived deterministically, their values are complex outputs of operations on a dynamically changing graph using secret path r . Without knowledge of r and $SKEPt$, these values provide no direct information about the secret key.
- **Path Unobservability:** The "Symbolic Mutation Security Model" (implied by file 26EchoPulse.Symbolic.Mutation.Security.Model.pdf and its connection to 24EchoPulse.SymbolPath.Coverage.Testing.pdf) aims to ensure that the symbolic path r is unobservable to adversaries. Even if v_enc were leaked, without r and the proper G_t context, an adversary cannot derive $KEPt$.
- **HORM Output ($hlocal$):** The $hlocal$ values from the HORM are intermediate hash outputs. Their combination into the final KDF input ensures that no single $hlocal$ reveals significant information about the path or the key. The use of strong, standard hash functions (like SHA256 or BLAKE2s) contributes to their cryptographic security.

- **Constant-Time Implementation:** As mentioned in Section 3.3, all operations, including the generation and processing of intermediate values, are designed to be constant-time to resist timing and power analysis attacks that could otherwise leak information from these values.

5. Conclusion

The EchoPulse PQ Forward Secrecy Key Derivation Model (v3.0) provides a robust and innovative approach to achieving post-quantum forward secrecy without relying on traditional ephemeral Diffie-Hellman constructions. By intrinsically binding the key derivation (KEPt) to a unique, time-bound symbolic graph state (Gt) through deterministic mutation ($\mu(G,t)$) and leveraging the Hash Oracle Replacement Model (HORM), EchoPulse ensures strong resistance to key compromise, replay attacks, and cross-session key prediction. Its design, optimized for minimal resource consumption and constant-time operations, makes it uniquely suitable for highly constrained embedded environments critical for future-proofing IoT, defense, and long-term communications against the imminent quantum threat. The detailed integration with existing EchoPulse v1.0-v2.1 mechanisms solidifies its position as a cohesive and ready-for-deployment PQ-FS solution.