EchoPulse Failure Case Handling (Document A3)

Version: 1.0

Date: May 11, 2025

Author: EchoPulse Initiative

This document defines the standard operational responses of the EchoPulse decapsulation function when encountering conditions that prevent the successful derivation of the shared secret key K' using the provided secret key (SK) and the received random symbol sequence (r). The primary objective is to ensure forward recovery from transient errors without compromising the security of the underlying symbolic graph structure or revealing sensitive state transition information to potential adversaries.

1. Purpose

This specification aims to:

* Define deterministic and secure reactions to decapsulation failures, specifically when the application of SK and r does not lead to the expected shared secret K'.

* Ensure a mechanism for forward recovery from temporary desynchronization or data corruption without exposing details of the graph's internal structure or transition function.

2. Failure Modes

The EchoPulse decapsulation function may encounter the following failure modes:

* **Mutation Desync:** This occurs when the receiving device's local state transition graph $G(V, E)$ is out of synchronization with the sender's graph due to a drift in the `session_index` or a loss of the correct mutation seed.

* **Invalid r:** The received random symbol sequence r leads to one or more undefined transitions when applied to the public key-derived state within the receiver's local $G(V, E)$.

* **Graph Drift:** The locally stored $G(V, E)$ has been unintentionally overwritten, corrupted, or is inconsistent with the expected state for the current session.

* **Replay Detection:** The received r is identical to one used in a previous session under a different mutated graph instance ($\mu(G, \text{salt}, \text{session\_index})$), indicating a potential replay attack.

3. Fallback Mechanism

Upon initial decapsulation failure, the following deterministic retry mechanism shall be engaged:

* **Retry Count:** The decapsulation process will be retried up to a maximum of 3 times.

* **Deterministic r’ Derivation:** For each retry attempt ($retry\_count = 1, 2, 3$), a new random symbol sequence r' will be derived using the following deterministic function:

```
r' = H(session_index || salt || retry_count)
```

where `session_index` is the current session identifier, and `salt` is the shared secret salt used for graph mutation.

* **New Transition Path:** Each derived r' will initiate a new and distinct traversal path through the local $G(V, E)$ from the public key-derived state.

* **Abort on Failure:** If all three retry attempts with the derived r' sequences fail to produce a valid shared secret K', the decapsulation process will be aborted, and an error status will be returned.

4. Graph Reinitialization

If the decapsulation retries fail, indicating a more significant desynchronization issue, the following graph reinitialization procedure should be initiated:

* **Fallback Mutation:** The local $G(V, E)$ will be recomputed using a fallback salt value (if provisioned) or the initial configuration parameters.

* **New Session Seed Derivation:** A new seed for the mutation function μ will be derived using the following function:

```
μ_seed = H(device_id || failure_count || last_v)
```

where `device_id` is a unique identifier for the device, `failure_count` is a persistent counter tracking the number of consecutive decapsulation failures exceeding the retry limit, and `last_v` is the final state reached during the last failed decapsulation attempt. This mechanism aims to prevent attackers from intentionally forcing predictable graph resets by inducing repeated failures.

* **Increment Failure Counter:** The `failure_count` will be incremented and persistently stored after each failed reinitialization attempt.

5. Decapsulation Timing Constraints

All retry attempts and fallback graph reinitialization operations must be completed within a fixed, pre-defined execution time window. This constraint is crucial to prevent timing-based side-channel attacks that might allow an adversary to infer information about the failure mode or the underlying graph structure based on the time taken for decapsulation. Implementations must be carefully engineered to meet these timing constraints.

6. Implementation Notes

The following implementation guidelines must be adhered to:

* **Non-Persistent Fallback State:** The fallback mechanisms (retrying with r') must not modify the persistent secret key (SK) or the core graph structure stored for the current session.

* **Non-Persistent Retry Counter:** The counter used to track the number of retry attempts ($retry\_count$) should be non-persistent and reset at the beginning of each new decapsulation attempt.

* **r’ Non-Reuse:** The derived r' values used for retries within a single decapsulation attempt must not be reused across different decapsulation sessions.

7. Symbolic Example (Optional)

Consider a scenario where:

* $SK = \text{0x7D B4 11 3C...}$

* Received $r = \text{0xF1 00 9A...}$ leads to decapsulation failure.

The following retry attempts would be made:

* **Retry 1:** $r'_1 = H(\text{0x05} || \text{salt} || \text{0x01})$. Applying SK and r'_1 leads to a final state v' and successful derivation of K'.

* If Retry 1 failed, Retry 2 would use $r'_2 = H(\text{0x05} || \text{salt} || \text{0x02})$, and so on, up to three retries.

If all retries fail, the graph reinitialization procedure outlined in Section 4 would be invoked.

8. Conclusion

The EchoPulse protocol incorporates a deterministic and secure failure handling mechanism for the decapsulation process. By employing a limited number of retries with dynamically derived random symbol sequences and a carefully controlled graph reinitialization procedure, EchoPulse aims to maintain forward recovery without leaking sensitive information about the symbolic graph structure or being susceptible to timing-based side-channel attacks.

Document A3 — Symbolic Recovery Layer — EchoPulse Initiative