

EchoPulse – Side Channel Defense Principles (v3.0)

1. Threat Model

Side-Channel Attacks (SCAs) pose a significant threat to cryptographic implementations, particularly in resource-constrained embedded systems. Unlike cryptanalysis, SCAs exploit physical leakage from a computing device (e.g., power consumption, electromagnetic radiation, timing variations, acoustic emissions, cache access patterns) to infer secret information. For EchoPulse, this means an adversary with physical access or close proximity could potentially:

- **Infer Symbolic Path (r):** By observing power traces or timing of graph traversals, infer which symbols in r_i are being processed or which edges/nodes are accessed.
- **Infer Internal Graph State (G_t):** Leak information about the dynamically evolving G_t or the mutation function $\mu(G, t)$.
- **Recover hlocal outputs:** Deduce intermediate Hash Oracle Replacement Model (HORM) outputs, which contribute to the final session key.
- **Predict forward_entropy_epoch (t):** Learn patterns in the derivation of t to compromise PQ-FS.
- **Distinguish Operations:** Differentiate between "correct" and "incorrect" path traversals or key derivations, aiding fault injection or brute-force.
- **Compromise Long-Term Keys:** Extract SKEP or SKLT through persistent leakage.

This document outlines a modular architecture to protect EchoPulse's core KEM, Multi-Party Key Exchange (MPKX), and Forward Secrecy (FS) layers against these pervasive SCAs.

2. Mitigation Strategy Overview

EchoPulse's SCA mitigation strategy is multi-layered, combining design-time algorithmic principles with runtime implementation techniques, tailored for resource-constrained environments. The core approach is to make operations independent of secret values, inject noise, and randomize execution flows.

Mitigation Matrix:

Attack Type	Core Principle	EchoPulse Method	Location(s)
-------------	----------------	------------------	-------------

Timing Analysis	Constant-Time Operations	Uniform KDF/HORM traversal; Fixed-time loops	KEM, HORM, Key Schedule, MPKX aggregation
Power/EM Profiling	Data-Independent Execution	Layout randomization; r_i fuzzing; Noise injection	HORM, Memory Access, Key Derivation
Memory Access (Cache)	Deterministic Access Patterns	Fixed-size buffers; Oblivious access	Graph Traversal, Mutation Buffers
Fault Injection	Redundancy; Integrity Checks	Pre/Post-computation checks; Epoch re-validation	Key Derivation, MPKX contribution verification
Acoustic Analysis	Data-Independent Execution Timing	Same as Timing/Power	All cryptographic operations
Differential Analysis	Entropy Masking; Output Obfuscation	$\mu'(G,t)$; Randomized symbol gates	HORM, Graph Mutation, Key Derivation

3. Timing / EM / Cache Defense Logic

3.1. Physical Security Architect: Symbolic Execution Model for SCA Resilience

To achieve robust SCA resilience, EchoPulse employs a symbolic execution model that aims to make cryptographic operations appear indistinguishable to an attacker, regardless of the secret values being processed.

- **r_i Rotation Fuzzing:**

- Instead of a fixed, secret symbolic path r_i , during path traversal, r_i is treated as a secret input to a "rotator" or "shuffler" function.
- The effective symbol s' for HORM at each step is not directly s_j from r_i , but $s_j \oplus \text{fuzz_value}_j$, where fuzz_value_j is derived from a public, but unpredictable, session-specific seed. The actual path in G_t remains deterministic from r_i , but the *computation* of s' appears randomized.

- This fuzzing introduces noise in power/EM traces, obscuring the actual symbolic path.
- **Uniform KDF/HORM Traversal Path Logic:**
 - All KDF and HORM operations *MUST* take a fixed amount of time and follow identical memory access patterns, irrespective of the values being processed (e.g., node IDs, symbol values, hash results).
 - This implies fixed-loop iterations for graph traversal, even if a condition could allow for early exit (e.g., "symbol not found" must still traverse all possible lookup paths).
 - For instance, if a node lookup involves comparing a value, all comparisons should be executed, and the result masked, rather than short-circuiting.
- **Entropy Masking with Randomized Symbol Gates:**
 - During HORM operations, the input symbol s_j is processed through a "symbol gate" that applies random masking (XOR with a secret, session-specific nonce) before actual graph lookup or hashing.
 - The result is then unmasked. This obscures the exact symbol value in power traces.
 - The same principle applies to intermediate values: $\text{masked_value} = \text{secret_value} \oplus \text{random_mask}$. All computations use masked values, and unmasking occurs only at the very end or for final outputs.
- **Layout Randomization Techniques:**
 - Memory addresses for graph nodes, edges, or lookup tables are randomized per session or periodically. This makes it harder for cache-timing attacks to infer access patterns, as the cache line used for a specific node/edge will vary.
 - This involves dynamic re-mapping or using randomized offsets for data structures. For example, Gt could be stored with a random offset in RAM that changes per session.

3.2. Constant-Time Cryptographic Engineer: Vulnerable Points & Mitigations

Critical cryptographic operations must be implemented in constant-time.

- **Vulnerable Points:**
 - **Graph Traversal (KEM, MPKX):** The iteration over the symbolic path r_i and the subsequent node/edge lookups. If the time to find a symbol or traverse an edge varies based on its value or position, it leaks information.
 - **HORM Logic:** The internal hashing (e.g., SHA256 compression function) and the logic for computing v' and h_{local} .

- **KDF (Key Derivation Function):** Any conditional logic or variable-time operations within the KDF that depend on secret inputs.
- **Shared Secret Comparison (e.g., v_enc validation in decapsulation):** If the comparison of vdec to venc_id takes variable time based on the number of matching bytes, it creates a timing oracle.
- **Mutation Function $\mu(G,t)$:** If the time taken to apply mutation varies based on the type or extent of the change, it could leak t.
- **Constant-Time Mitigations:**
 - **Fixed-Loop Iterations:** All iterations over symbolic paths (ri) and internal loops within HORM/KDF *MUST* run for a fixed, pre-determined number of cycles, regardless of the input value or outcome.
 - **Masked Operations:** Use bitwise operations (AND, XOR, OR) with masks instead of conditional branches (if/else) that depend on secret data. Example: result = (A & mask) | (B & ~mask) instead of if (condition) result = A; else result = B;.
 - **Oblivious Array Access:** When accessing arrays or lookup tables based on secret indices, use an oblivious read mechanism (e.g., read all possible elements and then select the correct one using a mask).
 - **Constant-Time Comparison:** Use a fixed-time comparison function (e.g., a bit-wise XOR and check for zero, always iterating through all bytes) for comparing secret values like vdec and venc_id.
- **Noise Injection Points:**
 - Inject dummy operations (e.g., NOPs, dummy memory writes/reads) at random intervals during graph traversal or HORM computation to obscure the timing characteristics.
 - Randomly vary the instruction fetch order (if supported by hardware) or insert randomized branches to constant targets.
- **Trap Fencing (Symbolic Stalling):**
 - If a detected anomaly (e.g., a suspected timing deviation, an unexpected memory access pattern) indicates a potential entropy leak, the protocol *MAY* enter a "symbolic stalling" state.
 - In this state, the device performs a fixed number of dummy HORM operations or graph traversals with public/dummy inputs for a randomized duration, while potentially attempting to re-establish a secure state or triggering an alarm. This makes it harder for the attacker to pinpoint the exact leakage.
- **HORM-Integrated Obfuscation:**
 - The HORM's internal logic itself can incorporate obfuscation. For instance,

the mapping from (v,s) to (v',hlocal) could involve a redundant, randomized computation path that always produces the same output but has varying intermediate steps.

- **Symbolic Entropy Multipliers:** $\mu'(G,t) = \mu(G,t) \oplus \epsilon(t_i)$
 - To enhance resistance against differential power analysis (DPA) targeting the mutation function, a small, session-specific, and pseudo-random "entropy multiplier" $\epsilon(t_i)$ is XORed (or otherwise combined) with the output of the mutation function $\mu(G,t)$ just before it's used for the KEM operation.
 - $\epsilon(t_i)$ is derived from the forward_entropy_epoch (t) and a secret session key component (known only to the device), and varies slightly over sub-epochs (t_i). This introduces variability in the actual graph used for computation, confusing DPA attacks. The $\epsilon(t_i)$ must be designed such that it does not alter the graph structure in a way that breaks KEM functionality, perhaps by operating on redundant or "dummy" graph elements.

4. Symbol Layer Randomization

The symbolic nature of EchoPulse offers unique opportunities for SCA defense.

- **Randomized Symbol Gates:** As described in 3.1, symbols are processed through gates that apply temporary, secret-dependent masks, hiding their true value from physical observations.
- **Symbol Aging and Volatility Logic:**
 - To counteract long-term statistical attacks (e.g., differential power analysis across many sessions), the mutation_schedule_id can specify policies for "symbol aging."
 - Symbols that have been heavily used or are prone to leakage (if statistically identified) can be made "volatile" – meaning their properties or positions on the graph are prioritized for randomization or mutation by $\mu(G,t)$ more frequently. This increases the entropy drift specifically for frequently used symbols, preventing a build-up of statistical information.
 - This helps in constantly changing the "attack surface" presented by the symbolic graph.

5. Embedded Systems Resilience Designer: Lightweight Runtime Defense

For microcontrollers and IoT devices with stringent constraints (RAM < 32KB, CPU < 40MHz, no persistent disk access), the defense mechanisms must be highly optimized.

- **EM Shielding Guidelines:**

- While not a software fix, hardware EM shielding (e.g., metal enclosures, ground planes, specific PCB layout) is a fundamental prerequisite for high-security IoT deployments. Software design *assumes* reasonable physical protection.
- **On-Device Key Refresh Cadence:**
 - Long-term keys are *never* stored on-device in plaintext. They are loaded securely and used for specific operations (e.g., derived for a short lifespan, then erased).
 - Ephemeral secrets like r_i and intermediate HORM states are generated freshly for each session and immediately wiped from RAM after use.
 - The `forward_entropy_epoch (t)` itself acts as a soft key refresh mechanism by changing the graph.
- **Side-Channel Aware Replay Detection Fallback:**
 - The `ReplayToken` validation (File 47) is already designed to be cryptographically robust.
 - For SCA-aware replay detection, the comparison of `ReplayToken'(P_i)` with `ReplayToken(P_i)` *MUST* be done in constant time.
 - If a replay is detected and SCA is suspected (e.g., via runtime integrity checks or anomaly detection), the `MPKX_Repair()` logic (File 47) should prioritize an immediate full state reset and regeneration of a *new* `forward_entropy_epoch` to ensure maximal entropy for the next attempt.
- **Symbol Aging and Volatility Logic (Runtime):**
 - The mutation function $\mu(G,t)$ can have parameterized options for symbol volatility. These parameters are fixed at compile-time or by `mutation_schedule_id`.
 - The runtime simply executes the $\mu(G,t)$ as defined, which inherently incorporates the symbol aging rules, thus not adding runtime overhead beyond the mutation itself.
- **Constraints Compliance:**
 - **RAM < 32KB:** All proposed SCA mitigation techniques are designed to operate within fixed, small memory footprints. Layout randomization does not imply massive memory duplication. Constant-time operations typically involve small, fixed-size buffers.
 - **CPU < 40MHz:** The emphasis is on simple, repetitive operations (fixed loops, bitwise masks) that are efficient on low-clock microcontrollers. Complex operations like true random number generation (TRNG) are offloaded to dedicated hardware (if available) or pre-seeded PRNGs.

- **No Persistent Disk Access:** All session-specific data (e.g., r_i , intermediate HORM states) are held in volatile RAM and immediately erased. Long-term keys are handled with care, potentially via secure elements or one-time loading.

6. Meta Role – Integration Blueprint

The SCA defense principles are designed for modular, layered integration into the EchoPulse ecosystem.

- **Compatibility with Files 43-47:**

- **File 43 (PQFS Structure):** The derivation of `forward_entropy_epoch (t)` and `Gt` is preserved. SCA defenses ensure this derivation itself is robust and doesn't leak `t`.
- **File 44 (PQFS Key Derivation Model):** The constant-time implementation of HORM operations and KDFs directly enhances the security of the `KEPt` derivation.
- **File 45 (TLS Handshake Extension):** The handshake process itself must be SCA-resistant, ensuring random number generation and parameter negotiation are robust.
- **File 46 (MPKX Architecture):** MPKX operations, particularly the generation of r_i and the computation of C_i , benefit directly from constant-time and randomization techniques.
- **File 47 (Replay Validation & Repair):** The `ReplayToken` generation and validation must be SCA-resistant to prevent leakage during the replay detection process.

- **Terminology and Interface Continuity:** All core EchoPulse terms ($\mu(G,t)$, `ReplayToken`, `KDF_HORM`, `entropy_epoch`, `v_enc`, `r_i`, `FS_token` conceptually as the `forward_entropy_epoch`) are consistently used. The concept of `FS_token` here refers to the underlying `forward_entropy_epoch (t)` which drives the dynamic aspect of FS.

- **Modularity for Layered Integration:**

- Each SCA mitigation technique (e.g., r_i rotation fuzzing, entropy masking, constant-time comparisons) can be implemented as a distinct, toggleable module.
- This allows for configurable security levels based on the threat environment and device capabilities (e.g., `SCA_HIGH`, `SCA_MEDIUM`, `SCA_LOW` profiles).
- The core cryptographic primitives (HORM, KDF, graph traversal) are designed such that SCA defenses are wrapped around them, rather than deeply

embedded, facilitating auditability and future updates.

This document provides a comprehensive framework for securing EchoPulse against side-channel attacks, ensuring its robust performance and confidentiality even in physically exposed or resource-constrained environments.