## Hash Oracle Substitution and ROM Integrity in EchoPulse: SHA3-256 vs. Lightweight Alternatives

### 1. Introduction

The EchoPulse Key Encapsulation Mechanism (KEM) relies on a cryptographic hash function, H, to derive the shared secret key K=H(venc||r), where venc is the final graph node and r is the symbolic payload. In its initial security proofs, H is modeled as a Random Oracle (ROM). While SHA3-256 serves as the baseline, its computational demands, particularly on resource-constrained embedded platforms like ARM Cortex-M0+ and RISC-V M-class microcontrollers, necessitate the evaluation of lightweight cryptographic hash alternatives. This document formally defines the properties required for H in the ROM, analyzes the suitability and implications of substituting SHA3-256 with alternatives (BLAKE2s, SHAKE128, Prehash + AES), and proposes a modular abstraction layer for managing these substitutions while preserving ROM integrity.

### 2. Hash Oracle Formalism (Hash Oracle Formalist)

In the Random Oracle Model (ROM), a hash function H:{0,1}∗→{0,1}L is idealized as a truly random function. This implies specific properties crucial for cryptographic security proofs.

- **Properties of an Ideal Random Oracle:**
  - **Collision Resistance:** For any distinct inputs x1,x2, it is computationally infeasible to find x1o=x2 such that H(x1)=H(x2).
  - **Preimage Resistance:** For any given output y, it is computationally infeasible to find an input x such that H(x)=y.
  - **Second Preimage Resistance:** For any given input x1, it is computationally infeasible to find x2o=x1 such that H(x2)=H(x1).
  - **Pseudorandomness:** The output of H on any input x appears to be uniformly random, even to an adversary with knowledge of other (xi,H(xi)) pairs, provided x has not been queried.
  - **Programmability (Idealized Behavior):** In the context of security proofs, a simulator can "program" the random oracle by defining the output for specific inputs. If a query for a new input x is received, the simulator generates a random output y and stores (x,y). If a query for an existing input x

is received, the simulator returns the previously stored y.

- Real-World vs. Ideal Random Oracle:
  No real-world hash function is a perfect random oracle. The ROM is a heuristic model that simplifies proofs by abstracting away the complexities of concrete hash function constructions. A proof in the ROM demonstrates that a protocol is secure if its hash function behaves like a random oracle.
- **Suitability of Alternatives in ROM Context:**
  - **BLAKE2s:** As a modern cryptographic hash function designed with efficiency and strong security guarantees, BLAKE2s is widely considered to be "ROM-friendly." Its properties (collision resistance, pseudorandomness) are strong enough to justify its treatment as a random oracle in security reductions, much like SHA3. It is often preferred over SHA3 in high-performance contexts due to its internal structure optimized for modern CPUs.
  - **SHAKE128:** As an eXtendable Output Function (XOF) based on the Keccak (SHA3) sponge construction, SHAKE128 offers variable output length. When used to produce a fixed 256-bit output (as in EchoPulse's LK), it functions similarly to SHA3-256 in terms of cryptographic properties. Its security relies on the soundness of the Keccak permutation. It can be safely modeled as a Random Oracle.
  - **Implications for Theoretical Reductions:** Replacing SHA3-256 with BLAKE2s or SHAKE128 in the ROM proof does not fundamentally alter the structure of the reduction. The "Hash Oracle" OH in the game remains the same abstract ideal. The implications arise if the *real-world* properties of the chosen hash function are found to deviate from the ROM ideal in a way that is exploitable by an adversary.

### 3. Embedded Hash Implementation Analysis (Embedded Hash Implementation Analyst)

Selecting a hash function for embedded systems involves critical trade-offs concerning performance, memory footprint, and constant-time behavior.

- **Comparison of Candidates:**

| Feature | SHA3-256 (Baseline) | BLAKE2s | SHAKE128 (256-bit output) | Prehash + AES (e.g., BLAKE2s + AES-256) |
|---|---|---|---|---|
| **Algorithm** | Keccak-based | HaS-1 based | Keccak-based | Hybrid: Stream |

| | | | | |
|---|---|---|---|---|
| | sponge, 24 rounds | (ChaCha-like), 10 rounds | sponge, 24 rounds (different capacity/rate) | cipher (prehash) + Block cipher (AES) |
| **Core Operation** | Permutation (Keccak-p) | G-function (parallel mixing) | Permutation (Keccak-p) | Permutation/ Bitwise ops (prehash) + Galois Field (AES) |
| **Code Size** | ~2-4 KB (software) | ~1-2 KB (software) | ~2-3 KB (software) | ~2-3 KB (prehash) + ~1-2 KB (AES software) |
| **RAM Usage** | ~200-256 bytes (state, buffer) | ~100-128 bytes (state, buffer) | ~100-128 bytes (state, buffer) | ~100-128 bytes (prehash) + ~100-200 bytes (AES state) |
| **Expected µs (Cortex-M0+)** | >1000 µs (for typical input, SW) | ~300-600 µs (faster than SHA3, SW) | ~500-900 µs (slower than BLAKE2s, faster than SHA3-256 SW) | ~200-400 µs (if AES HW accel, total) |
| **Constant-Time Suitability (SW)** | Challenging due to complex internal logic, lookup tables | Designed for constant-time, simpler operations | Challenging (similar to SHA3) | High (if AES HW is CT, and prehash is CT) |
| **Output Size** | Fixed 256-bit | Fixed 256-bit (for BLAKE2s) | Variable (specified length, e.g., 256 bits) | Fixed by AES block/mode |

- **Trade-offs:**
  - **Truncation Effects:** Truncating the output of any hash function (e.g., using

SHA3-256 to derive a 128-bit key) reduces the theoretical security strength to the truncation length. While improving speed, this must be acceptable within the system's security budget. The ROM proof would then implicitly assume a 128-bit Random Oracle.
    - **Fixed vs. Variable Output Size:** SHA3-256 and BLAKE2s provide fixed 256-bit outputs. SHAKE128 provides flexibility in output length, which is beneficial if variable key lengths are desired or if a smaller key is needed (e.g., 128-bit security for a specific scenario). For EchoPulse's LK=256, this is less of a differentiating factor.
    - **Entropy Retention:** All listed cryptographic hash functions are designed to retain high entropy from their inputs. The choice primarily impacts performance and implementation complexity on constrained devices.
    - **Hardware Acceleration:** If the target Cortex-M0+ (or more likely M4F/M33, or RISC-V with custom extensions) possesses AES hardware acceleration, the "Prehash + AES" approach becomes extremely attractive for its speed and inherent constant-time properties. The prehash function (e.g., BLAKE2s) would ensure the input to AES is sufficiently diffused.

**4. Security Reduction Implications (Security Reduction Technologist)**

The substitution of H directly impacts the security reduction from SGPU to IND-CCA2 in the ROM.

- Impact on SGPU-to-IND-CCA2 Reduction:
  The reduction proof constructs a simulator B that uses an IND-CCA2 adversary A to solve the SGPU problem. The crucial step involves B programming the Random Oracle H at the challenge phase. This step relies on the abstract property that H produces unpredictable outputs for distinct inputs, and consistent outputs for repeated inputs.
    - **No Fundamental Change:** If the replacement (BLAKE2s, SHAKE128) is indeed a strong cryptographic hash function behaving like a Random Oracle, the core logic of the reduction remains sound. The reduction does not depend on the specific *internal structure* of SHA3, but rather on its idealized Random Oracle properties.
    - **Proof Parameters:** The output length of H (i.e., LK=256) directly impacts the security strength. If this length is truncated, the λ (security parameter) of the proof should be adjusted accordingly. For instance, a 128-bit output would correspond to λ=128.
    - **Hash Collision Assumptions:** The proof relies on the assumption that an

adversary cannot find collisions for H (i.e., venc,1||r1=venc,2′||r1′ but venc,1||r1o=venc,2′||r1′ resulting in the same hash output). If the substituted hash function has known collision vulnerabilities, the ROM assumption is broken, and the proof fails. This is why using standard, well-vetted cryptographic hash functions is critical.

- **New Assumptions or Re-parameterization:**
  - No fundamentally *new* cryptographic assumptions are required if the chosen alternative (BLAKE2s, SHAKE128) is itself a cryptographically secure hash function that can be modeled as a Random Oracle.
  - However, the *parameterization* of the proof might need adjustment:
    - **Output Length (**LK**):** Must match the chosen hash function's output or the chosen truncation length.
    - **Computational Cost Modeling:** While the ROM abstracts computational cost, real-world benchmarks provide practical context for the feasibility of the scheme on target hardware.

**5. Model Abstraction Manager (Meta Role)**

To manage hash function flexibility and ensure consistent integration into EchoPulse, a modular abstraction layer is proposed.

- Modular Hash Abstraction (H_modular(input)):
  The EchoPulse KEM should interact with a generic HashFunction interface, allowing different backend implementations to be plugged in without altering the core KEM logic.

```
INTERFACE IHashFunction:
  METHOD Hash(input_bytes: ByteSequence) -> ByteSequence (length L_K)

CLASS SHA3_256_Backend IMPLEMENTS IHashFunction:
  METHOD Hash(input_bytes) -> SHA3_256(input_bytes)

CLASS BLAKE2s_256_Backend IMPLEMENTS IHashFunction:
  METHOD Hash(input_bytes) -> BLAKE2s(input_bytes)

CLASS SHAKE128_256_Backend IMPLEMENTS IHashFunction:
  METHOD Hash(input_bytes) -> SHAKE128_XOF(input_bytes, 256_bits)

CLASS Prehash_AES_Backend IMPLEMENTS IHashFunction:
  // Internal: prehash_alg: IHashFunction (e.g., BLAKE2s)
```

```
    // Internal: aes_block_cipher: IAES (e.g., AES-256 in CTR mode)
    METHOD Hash(input_bytes):
        prehashed_output = prehash_alg.Hash(input_bytes) // e.g., 256 bits
        // Use prehashed_output as key or IV for AES, then encrypt a fixed block (e.g., all zeros)
        // Or use AES in PRF-like mode.
        // Example: K = AES_CTR(Key=derived_aes_key_from_prehash, Nonce=fixed_nonce).Encrypt(ZeroBlock)
        Return AES_PRFLike_Mode(prehashed_output) // Output L_K bits
```

- Compatibility Flags & Modes:
  During compilation or system configuration, a flag can select the desired hash backend:
  - ECHO_PULSE_HASH_MODE = "SHA3_256_STANDARD"
  - ECHO_PULSE_HASH_MODE = "BLAKE2S_OPTIMIZED"
  - ECHO_PULSE_HASH_MODE = "SHAKE128_COMPACT"
  - ECHO_PULSE_HASH_MODE = "PREHASH_AES_HWACCEL"
- **Substitution Matrix:**

| **Backend (Concrete H)** | **Primary Use Case** | **ROM Compatibility Classification** | **Audit Recommendation** |
|---|---|---|---|
| **SHA3-256** | Baseline, interoperability | High | Verify software implementation for constant-time execution on target platforms. Focus on preventing side-channel leakage. |
| **BLAKE2s** | Performance, embedded opt. | High | Verify software implementation for constant-time (especially for loop unrolling, register usage). Benchmark on specific Cortex-M0+/RISC-V targets. |

| | | | |
|---|---|---|---|
| **SHAKE128** | Compactness, variable output | High | Similar to SHA3-256, verify constant-time properties. Benchmark against specific output lengths if different from 256 bits. |
| **Prehash + AES** | HW acceleration, high speed | High (assuming strong prehash & AES) | Thoroughly audit AES hardware integration (if applicable). Verify the chosen prehash function (e.g., BLAKE2s) is constant-time and sufficiently robust. Ensure the "AES-PRF-like" mode is cryptographically sound for key derivation and handles unique inputs correctly. |

**Audit Recommendations for Each:**

- **General:** All implementations should undergo thorough formal verification (e.g., using Frama-C, static analysis) for absence of critical bugs and adherence to constant-time properties, particularly for any look-up tables or conditional branches.
- **Hardware Integration:** For Prehash + AES, the interaction between software and hardware acceleration (e.g., register access, DMA) must be meticulously audited for side-channel vulnerabilities and correct functionality.
- **Performance Benchmarking:** Extensive benchmarking on actual target hardware is mandatory to confirm expected $\mu$s timings and memory footprints under realistic operating conditions (e.g., with different clock frequencies, compiler optimizations).

This modular approach ensures that EchoPulse can adapt to diverse hardware constraints without compromising the integrity of its security proofs based on the

Random Oracle Model.