
EchoPulse Hash Adapter Module: Embedded Hash Selection and Performance Benchmarking

1. Introduction

The EchoPulse Key Encapsulation Mechanism (KEM) requires a robust and efficient cryptographic hash function for shared secret derivation (K=H(venc||r)). While SHA3-256 serves as a foundational choice for its strong security properties and Random Oracle Model (ROM) compatibility, its performance on resource-constrained embedded platforms is often prohibitive. This document defines a modular H_adapter interface, analyzes the performance of alternative lightweight hash functions, and provides guidance for selecting the optimal hash backend for various embedded environments. The goal is to ensure that EchoPulse can be efficiently and securely deployed on microcontrollers ranging from ultra-low-power Cortex-M0+ to more capable RISC-V 64-bit embedded cores, maintaining cryptographic integrity within the ROM.

2. Hash Adapter Architecture (Hash Adapter Architect)

To ensure flexibility and maintainability, a modular H_adapter interface is designed to abstract the underlying hash function implementation.

2.1. H_adapter(input_bytes) Interface Design:

The H_adapter will conform to a simple, consistent interface:

C

```c
// H_adapter Interface
typedef struct {
    // Function pointer to the hash implementation
    void (*hash_function)(uint8_t* output, const uint8_t* input, size_t input_len);
    size_t output_length_bytes; // Fixed output length in bytes (e.g., 32 for 256-bit)
    const char* name;           // Name of the hash backend (for logging/identification)
    bool is_rom_compatible;     // Flag indicating suitability for ROM proofs
```

```
    bool is_constant_time;      // Flag indicating verified constant-time implementation
} H_Adapter_Config;

// Global instance or passed around
extern H_Adapter_Config current_H_adapter;

// Example usage:
// uint8_t derived_key[32]; // For 256-bit key
// uint8_t* hash_input = ...; // v_enc || r
// size_t hash_input_len = ...;
// current_H_adapter.hash_function(derived_key, hash_input, hash_input_len);
```

- **Accepts arbitrary byte input:** The input_bytes parameter can accommodate v_enc || r as a concatenated byte sequence.
- **Selectable Backends:**
  - **Compile-time selection:** Recommended for highly constrained devices where runtime overhead of dynamic dispatch is unacceptable. This involves preprocessor directives (#ifdef, #define) to include only the necessary hash implementation.
  - **Runtime selection:** For more capable platforms, a global pointer to H_Adapter_Config (or similar mechanism) could allow dynamic switching if required, though compile-time is generally preferred for security-critical functions.
- **Ensures deterministic and fixed-length output:** Each backend must guarantee that for the same input, the same fixed-length output is produced. For EchoPulse, this length is LK=256 bits (32 bytes).

2.2. ROM-Compliant vs. Non-ROM-Compatible Modes:

While all implemented cryptographic hash functions aim for strong pseudorandomness and collision resistance, the is_rom_compatible flag serves as a formal indicator based on community acceptance in ROM proofs:

- **ROM-Compliant:** True for SHA3-256, BLAKE2s, SHAKE128. These are generally accepted as suitable for modeling as Random Oracles in theoretical security proofs.
- **Non-ROM-Compatible:** Could be used for custom or experimental hash functions not yet vetted for ROM-compatibility, or if specific optimizations (e.g., highly truncated outputs without corresponding security analysis) render the ROM assumption invalid. For EchoPulse's core security proofs, only ROM-

compliant backends are to be used.

3. Embedded Hash Benchmarking (Embedded Hash Benchmark Engineer)

Benchmarking is conducted on representative embedded platforms using optimized software implementations. All measurements are for a fixed 256-bit output. Input size for v_enc || r is assumed to be 64 bytes (10-bit state venc padded to 4 bytes, 256-bit payload r is 32 bytes, sum of venc and r can be 36 bytes. We assume padding to block size, e.g., 64 bytes).

3.1. Benchmarking Methodology:

- **Platform Details:**
  - **Cortex-M0+:** Representative MCU (e.g., STM32L0 series). Low clock speed (e.g., 32 MHz), minimal instruction set, no FPU.
  - **Cortex-M4F:** Representative MCU (e.g., STM32F4 series). Higher clock speed (e.g., 100 MHz), DSP instructions, Floating Point Unit (FPU).
  - **RISC-V 64-bit Embedded Class:** Representative core (e.g., SiFive E76, U540). Assumed to have M (Integer Multiplication/Division) extension.
- **Measurement Metrics:**
  - **Runtime (µs):** Time taken to compute one 256-bit hash. Measured using precise hardware timers (e.g., DWT Cycle Counter on ARM, RISC-V M-mode timers). Average over 103 to 105 repetitions to minimize noise.
  - **RAM Usage (bytes):** Peak stack and heap usage during hashing.
  - **Flash/Code Size (bytes):** Binary size of the hash function implementation.
  - **Constant-Time (CT) Behavior:** Verified through static analysis (e.g., inspecting assembly for data-dependent branches/lookups) and dynamic timing analysis (e.g., varying input bits and observing execution time deviation). Reported as a boolean flag.

3.2. Benchmark Results (Hypothetical Data for Illustration):

Hash Backend	Platform	Runtime (µs)	RAM (bytes)	Flash (bytes)	Constant-Time (CT)
SHA3-256	Cortex-M0+	~1200	~256	~3500	No (Hard SW)
SHA3-256	Cortex-M4F	~350	~256	~3500	No (Hard

					SW)
SHA3-256	RISC-V 64-bit	~400	~256	~3000	No (Hard SW)
BLAKE2s	Cortex-M0+	**~450**	~128	**~1500**	Yes
BLAKE2s	Cortex-M4F	**~80**	~128	~1500	Yes
BLAKE2s	RISC-V 64-bit	**~110**	~128	~1200	Yes
SHAKE128 (256b)	Cortex-M0+	~800	~128	~2500	No (Hard SW)
SHAKE128 (256b)	Cortex-M4F	~200	~128	~2500	No (Hard SW)
SHAKE128 (256b)	RISC-V 64-bit	~250	~128	~2000	No (Hard SW)
Prehash+AES	Cortex-M0+ (No HW)	~1000	~200	~3000	No (SW AES)
Prehash+AES	Cortex-M4F (w/ HW)	**~30**	~200	~1800	**Yes (HW CT)**
Prehash+AES	RISC-V 64-bit (w/ Crypto Ext)	**~40**	~200	~1500	**Yes (HW CT)**

Note: Prehash+AES assumes BLAKE2s for prehash, and a hardware-accelerated AES module for Cortex-M4F and RISC-V Crypto Extensions. Otherwise, performance degrades significantly if AES is in software.

4. EchoPulse Integration Evaluation (EchoPulse Integration Evaluator)

The choice of hash backend has direct implications for EchoPulse's performance and security properties.

- **Impact on KeyGen Entropy:**
  - The hash function H maps v_enc || r to the final shared secret K. Its role is to ensure that K is uniformly random given a sufficiently random v_enc || r input.
  - As long as the chosen hash backend is a cryptographically secure hash function (e.g., SHA3-256, BLAKE2s, SHAKE128), and the input v_enc || r possesses sufficient entropy (guaranteed by the Symbolic Graph Path Unpredictability (SGPU) and random payload r), the derived key K will be a high-entropy value. No direct impact on entropy generation *from* the KEM itself, but on its *transformation* into the final key.
- **Impact on Encapsulation Time:**
  - The hash computation K=H(venc||r) is the final step of the encapsulation process. Its performance directly adds to the total encapsulation time.
  - As evidenced by the benchmarks, using a lightweight alternative like BLAKE2s or a hardware-accelerated Prehash+AES can drastically reduce encapsulation time, making EchoPulse feasible on very constrained devices where SHA3-256 would be a bottleneck.
- **Impact on SGPU Unpredictability if Backend Hash Changes:**
  - The SGPU assumption (Symbolic Graph Path Unpredictability) is independent of the specific hash function used. SGPU relates to the hardness of predicting graph paths, which are generated *before* hashing.
  - Changing the hash backend **does not** impact the SGPU assumption itself, provided the new hash function maintains the necessary cryptographic properties (collision resistance, pseudorandomness) for the key derivation step.
  - If the backend hash *itself* becomes predictable or suffers from collisions, it could allow an adversary to deduce venc||r from K, or force collisions in K. This would then break the overall KEM security, regardless of SGPU. Therefore, the hash function must remain strong.
- **Security Properties of Prehash + AES:**
  - The security of Prehash + AES relies on two main pillars:
    1. **Prehash Strength:** The prehash (e.g., BLAKE2s) must be collision-resistant and output sufficiently random values to serve as a strong input for AES.
    2. **AES as a Pseudorandom Function (PRF)/Permutation:** AES, when used

correctly in a mode like CMAC, CTR, or a PRF construction, provides strong pseudorandomness and is well-analyzed.

- If AES hardware acceleration is present and side-channel resistant, this method can be highly secure and performant. Its security is generally considered robust and ROM-compatible in its entirety. The challenge is ensuring the correct and secure instantiation of the AES mode for key derivation, as well as the prehash's properties.

5. Result Summary and Recommendations (Result Coordinator)

5.1. Platform-by-Hash Performance & Properties:

Table 1: EchoPulse Hash Backend Benchmarks (Hypothetical)

Platform	Hash Backend	Runtime (µs)	RAM (bytes)	Flash (bytes)	Constant-Time (CT)	ROM Compatible
Cortex-M0+	SHA3-256	~1200	~256	~3500	No	Yes
	BLAKE2s	**~450**	**~128**	**~1500**	**Yes**	**Yes**
	SHAKE128 (256b)	~800	~128	~2500	No	Yes
	Prehash+ AES (SW)	~1000	~200	~3000	No	Yes
Cortex-M4F	SHA3-256	~350	~256	~3500	No	Yes
	BLAKE2s	**~80**	**~128**	**~1500**	**Yes**	**Yes**
	SHAKE128 (256b)	~200	~128	~2500	No	Yes

	Prehash+ AES (HW)	**~30**	~200	~1800	**Yes**	**Yes**
RISC-V 64-bit	SHA3-256	~400	~256	~3000	No	Yes
	BLAKE2s	**~110**	**~128**	**~1200**	**Yes**	**Yes**
	SHAKE128 (256b)	~250	~128	~2000	No	Yes
	Prehash+ AES (HW)	**~40**	~200	~1500	**Yes**	**Yes**

5.2. Security Alignment Matrix:

Hash Backend	SGPU Proof Alignment	ROM Proof Alignment	Practical Side-Channel Risk (SW)
SHA3-256	Full	Full	High (due to complex SW)
BLAKE2s	Full	Full	Low (designed for CT)
SHAKE128	Full	Full	High (due to complex SW)
Prehash + AES	Full	Full	Low (if HW-accelerated & CT)

5.3. Summary Recommendations per Device Class:

- **For Ultra-Constrained Cortex-M0+ Targets:**

    - **Recommendation: BLAKE2s.** It offers the best balance of speed, minimal RAM/Flash footprint, and crucial constant-time properties, making it the most suitable software-only solution for these devices.
- **For Cortex-M4F (with AES Hardware Acceleration):**
    - **Recommendation: Prehash + AES (e.g., BLAKE2s + AES-256).** This combination provides significantly superior performance due to hardware acceleration and maintains strong constant-time behavior.
- **For RISC-V 64-bit Embedded Class (with Crypto Extensions):**
    - **Recommendation: Prehash + AES (via Crypto Extensions).** Similar to Cortex-M4F, leveraging hardware extensions for AES will yield the highest performance and best side-channel resistance. If no crypto extensions are available, BLAKE2s is the strong software-only fallback.
- **General Recommendation:** Avoid direct software implementation of SHA3-256 or SHAKE128 on extremely constrained or side-channel-sensitive targets unless no alternatives exist or substantial assembly optimization with CT-guarantees is feasible.

5.4. Visual Representation: Runtime (µs) Heatmap

(Conceptual Heatmap Visualization - cannot be rendered directly in text)

A heatmap or bar chart illustrating the Runtime (µs) across platforms and hash backends, with darker/lower values indicating better performance, would visually reinforce the above table. BLAKE2s and HW-accelerated AES would consistently appear as the best performers in their respective categories.

6. Conclusion and Future Work

The H_adapter module enables EchoPulse to strategically select its hash backend based on target hardware capabilities, optimizing performance and resource utilization without compromising the integrity of its security proofs in the Random Oracle Model. BLAKE2s emerges as the strong candidate for software-only constrained environments, while hardware-accelerated AES with a robust prehash is optimal when available. Future work includes developing and formally verifying the constant-time reference implementations for each recommended hash backend on specific microarchitectures. This ensures that the theoretical ROM integrity translates into practical side-channel resistance.
