A Hash-Based AEAD Encryption Scheme for embedded Systems

Deepak Gauba
Security Architecture and Cryptographic Systems
Email: dgauba@gmail.com

Abstract

This paper presents a lightweight authenticated encryption with associated data (AEAD) scheme designed for embedded systems that lack hardware acceleration for block ciphers such as AES but provide support for SHA-256 or HMAC. The construction relies exclusively on HMAC-SHA256, using it both as a pseudorandom function (PRF) for keystream generation and as a message authentication code. Strong cryptographic key separation is enforced through a hierarchical derivation process inspired by HKDF.

A per-message pseudorandom key (PRK), derived from a master key and a fresh random initialization vector (IV), ensures that the master key is never used directly for encryption or authentication. The resulting design provides confidentiality, integrity, and authenticity while avoiding block-cipher dependencies entirely. Apart from these security advantages, most importantly, the construction provides quantum-robust security margins, offering a practical upgrade path for devices with SHA-256 hardware that would otherwise require costly hardware replacement to achieve post-quantum security.

The paper describes the design goals, threat model, key derivation hierarchy, encryption and authentication mechanisms and security considerations relevant to embedded implementations.

1. Introduction

Modern embedded devices increasingly incorporate low-power cryptographic accelerators that support hashing primitives such as SHA-256 or HMAC. However, many platforms—particularly low-cost microcontrollers and legacy SoCs—lack hardware support for AES or other block ciphers. In such environments, system designers face a difficult choice:

- Implement AES, purely in software, which is often slow, power-intensive, and prone to timing or side-channel attacks if not carefully engineered, or
- Adopt alternative cryptographic constructions based on hash functions or sponge primitives.

This work explores a practical and efficient AEAD construction entirely based on HMAC-SHA256, tailored for embedded systems with limited cryptographic hardware capabilities. Unlike conventional AEAD schemes such as AES-GCM, AES-CCM, or ChaCha20-Poly1305, the proposed design eliminates block ciphers altogether. Instead, HMAC-SHA256 is used as a deterministic PRF to generate a keystream in a counter-like fashion, while a separate HMAC invocation provides message authentication.

A fresh, random IV is required for each message. This IV is also used to derive, unique per-message, encryption and authentication keys without ever exposing or directly using the master key. The design follows well-established cryptographic principles, including encrypt-then-MAC composition, domain separation, and HKDF-style key derivation.

1.1 Post Quantum Considerations

While AES-128 remains secure against classical attacks but the advent of quantum computing introduces new threats.

The Grover's algorithm can reduce AES-128's effective security from 128 bits to 64 bits, rendering it potentially vulnerable in a post-quantum world. A migration to AES-256 would require hardware replacement at scale across billions of deployed devices, rendering the approach economically and logistically infeasible.

In contrast, SHA-256 offers inherent quantum resistance. Grover's algorithm reduces its effective security from 256 bits to 128 bits, maintaining an adequate security margin. Devices currently deployed with SHA-256 hardware accelerators can adopt this construction without hardware modifications, providing a practical path to post-quantum AEAD capabilities for existing infrastructure.

This future-proofing characteristic is particularly valuable for: - Long-lifecycle embedded systems (industrial IoT, automotive, infrastructure) - Devices with difficult or impossible firmware update paths - Systems where hardware replacement costs exceed the device value or may not be even possible.

2. Design Objectives

The construction is guided by the following objectives:

1. Eliminate reliance on AES or other block ciphers on platforms without hardware acceleration.
2. Provide confidentiality using a secure, hash-based stream cipher construction.
3. Provide integrity and authenticity using HMAC.
4. Ensure that the master key is never used directly for encryption or authentication.
5. Derive unique per-message keys using a fresh, random 256-bit IV.
6. Support associated data (AAD) that is authenticated but not encrypted.
7. Avoid length-extension and related attacks through fixed-size HMAC inputs.
8. Maintain conceptual simplicity and implementation feasibility for embedded systems.
9. Reduce susceptibility to side-channel attacks through uniform, data-independent operations.
10. Provide quantum-robust security margins suitable for long-lifecycle embedded deployments without requiring hardware changes.

3. Threat Model and Security Goals

Attacker Capabilities:

The attacker is assumed to:

- Observe IVs, ciphertexts, and authentication tags.
- Modify, replay, reorder, or inject messages.
- Perform chosen-plaintext and chosen-ciphertext attacks.

Security Goals:

The scheme aims to protect:

- Confidentiality of plaintext data.
- Integrity and authenticity of ciphertext and associated data.
- Secrecy of the long-term master key.

- Resistance to common side-channel attack vectors.

The construction targets standard AEAD security notions, including IND-CCA confidentiality and INT-CTXT integrity, under the assumption that HMAC-SHA256 behaves as a secure PRF.

4. Key Derivation Strategy

Key separation is fundamental to the design. Encryption and authentication per-message keys are derived from a pseudorandom key (PRK) generated using the master key and a fresh 512-bit IV. The PRK is then used to derive independent encryption and authentication keys through a misuse-resistant structural permutation of IV halves.

Key hierarchy:

```
IV = IV0 || IV1 (each 256 bits)
PRK    = HMAC(K_master, IV)
```

Notes:

- *HMAC internally pads the 32-byte* `K_master` *to 64 bytes (SHA-256 block size) as per standard HMAC construction.*
- *This padding is deterministic and safe; it does not impact message alignment goals.*
- *The IV is public and uniformly random; no internal structure is assumed and deliberately kept it such that it matches the SHA-256 input block size.*

This hierarchy ensures that compromise of any per-message key does not reveal the master key or other derived keys.

4.2 Pseudorandom Key (PRK)

```
PRK = HMAC(K_master, IV)
```

This step mirrors the *Extract* phase of HKDF:

- PRK is uniformly distributed and unique per message.
- The master key is never used beyond this derivation.
- IV entropy directly influences all derived keys.
- The IV is public and does not need to be secret.

4.3 Encryption and Authentication Key Derivation

```
B_enc = (~IV0)||(IV1)
B_auth = (IV0)||(~IV1)
```

Where ~ denotes bitwise NOT on each bit of the 256-bit half.

Then:

```
K_enc = HMAC(PRK, B_enc)
K_auth = HMAC(PRK, B_auth)
```

Properties:

- Both `B_enc` and `B_auth` are exactly 64 bytes, fully SHA-256 block-aligned.

- Structural permutations ensure that even if IV halves are accidentally reused, the encryption and authentication keys remain distinct.
- No ASCII labels or zero padding are required.
- Partial IV inclusion ensures domain separation.

This strict separation between `K_enc` and `K_auth` prevents cross-domain attacks, such as keystream recovery via MAC oracles.

5. Encryption (CTR-Like Stream Construction)

5.1 Keystream Generation

For counter value `i`:

```
KS[i] = HMAC(K_enc, IV_first_480_bits || counter_32_bits)
```

- Output size: 256 bits per block.
- Counter is a 32-bit big-endian integer.
- Blocks are generated sequentially until the plaintext is covered.

5.2 Encryption and Decryption

Encryption:

```
C[i] = P[i] XOR KS[i]
```

Decryption is identical. This construction is analogous to AES-CTR or ChaCha20, replacing the block cipher with an HMAC-based PRF.

6. Authentication

Authentication follows an encrypt-then-MAC paradigm:

The authentication tag is computed over the logically constructed input.

```
Tag = HMAC(K_Auth, AAD || Ciphertext || ZeroPadding || Length_Block)
```

Here:

```
Length_Block = AAD_Length || Message_Length
```

The Authentication data size is multiple of 64 bytes, and the block length encoding occupies 16 bytes, consisting of the AAD length and Message length encoded as two 64-bit unsigned integers in big endian.

To ensure deterministic placement of the length fields and to simplify implementation correctness, this construction enforces the following invariant:

The last 16 bytes of the authenticated data SHALL always contain the encoded AAD and Message lengths.

6.1 Padding Rules

Best Case Expansion:
If the remaining space in the last 64-byte block, after the ciphertext, is greater than or equal to 16 bytes, zero padding is applied such that the ciphertext is extended to exactly 48 bytes. The 16-byte length field (AAD_Length || Message_Length) is then appended, completing the final 64-byte block - 8 bytes for each.

Worst Case Expansion:
In the worst case, where the ciphertext length leaves less than 16 bytes available in the last block, this scheme appends up to 79 bytes of additional data:

- Up to 15 bytes of zero padding to complete the current block, and
- One additional 64-byte block containing the length encoding.
  - The first 48 bytes are set to zero, and
  - The final 16 bytes contain the encoded `(AAD_Length || Message_Length)`.

In the case when the cipher text exactly multiple if 64 bytes, another 64-byte block is still added, with 48 bytes of zeros and 16 bytes of length fields.
This construction guarantees that the length fields are always located at a fixed position at the end of the authenticated data, independent of the ciphertext length.

Design Rationale:
This padding strategy is intentional and differs from conventional hash-based padding schemes. By enforcing a fixed location for the length fields, the construction avoids ambiguity in multi-buffer authentication processing, reduces implementation complexity, and minimizes the risk of padding and length-handling errors in constrained or embedded environments.

7. Security Considerations

7.1 Key Separation

Independent derivation of `K_enc` and `K_auth` ensures that weaknesses in one domain do not compromise the other.

7.2 IV Requirements

IV reuse is catastrophic, as in any CTR-like construction. Implementations **MUST** ensure IV uniqueness:

- IVs must be generated using a cryptographically secure RNG.
- Uniform distribution is required; biased or repeated IVs must be rejected.

7.3 Master Key Protection

The master key is only used once per message to derive the PRK, reducing its exposure and attack surface.

7.4 HMAC Security Assumptions

HMAC-SHA256 is assumed to be a secure PRF. Under this assumption, both the keystream generator and MAC are cryptographically sound when used with strict key separation.

8. Implementation Guidance

- Counters are 32-bit big-endian values.
- Tag verification must be constant time.
- PRKs must never be reused across messages.
- Memory access patterns should avoid secret-dependent branching.
- Zero sensitive buffers after use when possible
.

9. Performance Considerations

SHA-256 produces 256-bit outputs, doubling the keystream block size compared to AES. On platforms with optimized hash implementations/accelerators, this can result in higher throughput than AES-CTR or AES-CCM. Performance is platform-dependent and should be evaluated accordingly.
Use of SHA-512 or SHA-3 variants may further increase throughput on systems that support wider hash outputs.

10. Future Work

Planned extensions include:

- Polynomial MACs like GHASH for parallel authentication.
- Explicit HKDF-style domain labels.
- Formal security proofs.
- Comparative benchmarks against AES-CTR, AES-CCM, ChaCha20, and hash-based DRBGs.

11. Formal Security Claims

11.1 Confidentiality

The scheme achieves IND-CCA confidentiality assuming HMAC-SHA256 is a secure PRF and IVs are never reused.

11.2 Integrity and Authenticity

The scheme achieves INT-CTXT integrity assuming HMAC-SHA256 is EUF-CMA secure.

11.3 Key Compromise Isolation

Compromise of a per-message key does not compromise other messages or the master key.

11.4 Non-Claims

- The scheme does not provide replay protection or forward secrecy across master key compromise.
- No claim against cryptographically relevant quantum computers
- No claim of NIST PQC compliance.

12. Review Against NIST SP 800-56

- Uses FIPS approved primitives.
- Follows HKDF-style extract-and-expand principles.
- Enforce domain separation and context binding.

The scheme does not claim compliance with asymmetric key establishment.

13. Positioning Statement

This construction is intended as a hash-based AEAD mechanism for constrained embedded systems, leveraging only FIPS-approved hash and MAC primitives. It aligns with NIST guidance on PRF-based key derivation and secure random number generation and is suitable for evaluation as a specialized AEAD profile.

A key advantage of this approach is its post-quantum resilience. While AES-128-based systems face a significant security reduction under Grover's algorithm (128→64 bits), SHA-256 maintains a 128-bit post-quantum security level—adequate for most embedded applications. This enables existing hardware platforms with SHA-256 accelerators to achieve post-quantum AEAD capabilities through firmware updates alone, avoiding the substantial costs of hardware replacement across deployed device fleets.

14. Conclusion

This paper presents a practical AEAD construction based entirely on HMAC-SHA256, suitable for embedded systems lacking AES hardware support. By enforcing strict key separation, deriving per-message keys, and following established cryptographic design principles, the scheme provides confidentiality and integrity comparable to block-cipher-based AEAD modes while remaining well-suited for constrained embedded environments.