Quantum-Safe Message Authentication Code – QMAC

Revision 1a, February 04, 2025

John G. Underhill – john.underhill@protonmail.com

This document is an engineering level description of the QMAC message authentication code generator. This document describes the mathematical foundations, secureity properties, and functions and usage of the QMAC MAC generator.

Contents **Page**

Foreword

This document is the first revision of the specification of QMAC, further revisions may become necessary during the pursuit of a standard model, and revision numbers shall be incremented with changes to the specification. The reader is asked to consider only the most recent revision of this draft, as the authoritative implementation of the QMAC specification.
The author of this specification is John G. Underhill, and can be reached at john.underhill@protonmail.com

1. Introduction

QMAC is a message authentication code (MAC) generator function designed to provide quantum-resistant integrity and authenticity guarantees. Structurally similar to GMAC, QMAC extends the field arithmetic from GF(2^128) to GF(2^256) and derives its subkeys using SHA3/SHAKE rather than AES. This document specifies the QMAC construction in detail. We provide full mathematical definitions of the underlying cmul256 (256-bit carry-less multiplication) and gfmul (Galois Field multiplication) functions, describe the state management functions (initialize, update, finalize, and dispose), and analyze the security of the scheme, including diffusion properties and post-quantum query bounds.

2. Mathematical Foundations

2.1 GF(2) and Polynomial Arithmetic

Let GF(2) be the binary field {0,1}with addition defined as XOR and multiplication defined as AND. An *n*-bit binary number

$$A = \sum_{i=0}^{n-1} \binom{n}{k} a_i \, x^i, a_i \, \epsilon \, \{0, 1\},$$

can be interpreted as a polynomial A(x) over GF(2).
Carry-less (or polynomial) multiplication of two such polynomials is defined as

$$R(x) = A(x) \cdot B(x) = \bigoplus_{\substack{i = 0 \\ n = 1}} (a_i \cdot (B(x) \ll i)),$$

where '$\ll i$' denotes multiplication by x^i (i.e. a left shift by i bits) and the sum is computed by XOR.

2.2 The 256-Bit Field and the Irreducible Polynomial

A 256-bit field element A is represented as

$$A(x) = \sum_{i=0}^{255} a_i \, x^i \, .$$

QMAC defines the finite field GF(2^256) using the irreducible polynomial

$$m(x) = x^{256} + x^{19} + 1$$

This means any polynomial $C(x)$ of degree up to 511 can be reduced modulo $m(x)$ by replacing any occurrence of x^{256} with $x^{19} + 1$. In particular, if the full product of two 256-bit elements is

$$R(x) = L(x) \oplus x^{256} \cdot H(x),$$

then the reduction is given by

$$R(x) \bmod m(x) = L(x) \oplus (H(x) \oplus (H(x) \ll 19)),$$

where '≪ 19' denotes a left shift by 19 bits with full carry propagation and appropriate folding of overflow.

2.3 The cmul256 Function

Let

$$A(x) = \sum_{i=0}^{255} a_i\, x^i\,, \qquad B(x) = \sum_{i=0}^{255} b_i\, x^i\,.$$

be two 256-bit elements. Their carry-less product is

$$R(x) = A(x) \cdot B(x) = \bigoplus_{i\,+\,j\,=\,k} (a_i b_j) xk, 0 \le k \le 510.$$

Decompose each 256-bit element into two 128-bit halves:

$$A(x) = A0(x) + x^{128}\, A1(x), \qquad B(x) = B0(x) + x^{128}\, B1(x).$$

Define the following 128-bit multiplications:

$$P0(x) = cmul128\big(A0(x), B0(x)\big),$$

$$P1(x) = cmul128\big(A1(x), B1(x)\big),$$

$$P2(x) = cmul128(A0(x) \oplus A1(x), B0(x) \oplus B1(x)).$$

Then the cross term is

$$M(x) = P0(x) \oplus P1(x) \oplus P2(x).$$

Reassemble the full 512-bit product R(x)R(x)R(x) as four 128-bit words: P_0

$$R_0(x) = \text{Low}_{128}\{\, P_0(x)\,\},$$

$$R_1(x) = \text{High}_{128}\{P0(x)\} \oplus \text{Low}_{128}\{M(x)\},$$

$$R_2(x) = \text{High}_{128}\{M(x)\} \oplus \text{Low}_{128}\{P1(x)\},$$

$$R_3(x) = \text{High}_{128}\{P1(x)\}.$$

Thus, mathematically,

$$R(x) = R0(x) \oplus x^{128}\, R1(x) \oplus x^{256}\, R2(x) \oplus x^{384}\, R3(x).$$

2.4 The gfmul Function

Given the 512-bit product $R(x) = L(x) \oplus x^{256} \cdot H(x)$, reduce it modulo

$$m(x) = x^{256} + x^{19} + 1.$$

Since

$$x^{256} \equiv x^{19} + 1 (\mathrm{mod}\, m(x)),$$

substitute to obtain

$$R(x) \bmod m(x) = L(x) \oplus (H(x) \oplus (H(x) \ll 19)).$$

Thus, the field multiplication is defined by:

$$\mathrm{gfmul}(A, B) = L \oplus (H \oplus (H \ll 19)).$$

The left shift is performed with full carry propagation. In the implementation, any bits shifted out beyond the 256-bit boundary are folded back (using XOR) into the lower part as required by the equivalence.

3. Comparison with GMAC

GMAC is defined over GF(2^128) using an irreducible polynomial (e.g. $x^{128} + x^7 + x^2 + x + 1$). Its multiplication computes a 256-bit product that is then reduced modulo the polynomial to yield a 128-bit result. QMAC, in contrast, uses 256-bit operands:

- **Operand Size:** QMAC operates over GF(2^256) versus GF(2^128) for GMAC.
- **Intermediate Product:** QMAC produces a 512-bit product.
- **Reduction:** QMAC reduces using the relation $x^{256} \equiv x^{19} + 1$ (versus a relation derived from a degree-128 polynomial for GMAC).
- **Subkey Derivation:** QMAC derives its subkeys via SHAKE (or cSHAKE) instead of AES, which is believed to be more quantum-resistant.

QMAC offers improved diffusion and a larger output tag, making it suitable for applications requiring stronger guarantees and post-quantum security.

4. Security Analysis

4.1 Diffusion and Linear Properties

The cmul256 operation is linear over GF(2) but, when used with secret subkeys and XORing in message blocks, forms an almost universal hash. The Karatsuba-based construction ensures that every bit of the 256-bit operands influences many bits in the 512-bit product, yielding strong diffusion.

4.2 Security Bounds

- **Classical Security:**
  With a 256-bit output, the birthday bound implies collision resistance on the order of 2^{128} queries.
- **Quantum Security:**
  Using Grover's algorithm, the effective security level is roughly halved. Thus, if subkeys are derived via SHAKE-256, the effective bound is approximately 2^{128} operations classically and 2^{64} quantum queries.
- **SHAKE-512 Variant:**
  If SHAKE-512 is used for subkey derivation, then the effective security level rises to approximately 512 bits classically and 256 bits against quantum adversaries, thereby drastically reducing the effectiveness of quantum attacks.

5. Implementation: AVX/AVX2 and Scalar Versions

5.1 AVX/AVX2 Implementation Using __m256i and __m128i

The AVX2 implementation uses 256-bit registers (__m256i) to load, store, and operate on 256-bit field elements directly. The core functions are:

- **avx_clmul128:**
  Uses four calls to _mm_clmulepi64_si128 to multiply two 128-bit operands and produce a 256-bit result.
- **qsc_memutils_clmulepi64_si256:**
  Multiplies two 256-bit elements (each as 4 uint64_t's) to produce an 8-word (512-bit) product via a Karatsuba method.
- **qmac_shift256_left_19:**
  Shifts a 256-bit __m256i value left by 19 bits with proper carry propagation, folding the final carry into the least-significant word.
- **qmac_gfmul256_poly19:**
  Splits the 512-bit product into lower and upper 256-bit halves L and H, computes H ≪19, and reduces via:

$$r = L \oplus (H \oplus (H \ll 19))$$

In AVX implementations using __m128i, 256-bit values are represented as two __m128i registers. The shifting and reduction functions are re-implemented to work on arrays of four 64-bit words using SSE intrinsics (e.g. qmac_shift256_left_19_m128i).

5.2 Scalar Implementation

In the scalar implementation, 256-bit numbers are stored as arrays of 4 uint64_t's. The full 512-bit product is stored as an array of 8 uint64_t's, and the reduction is performed in a 320-bit container (an array of 5 uint64_t's) to capture overflow when shifting left by 19 bits. This ensures that every bit is processed identically in both implementations.

6. QMAC Functions

6.1 Initialization: `qsc_qmac_initialize`

- **Input:** A key K (256 bits), a nonce N (32 bytes), and an optional info string I.
- **Operation:**
  1. A cSHAKE (or SHAKE) instance is initialized with K, N, and I.
  2. Two 256-bit subkeys are "squeezed" from cSHAKE: the first 256 bits form H (hash subkey) and the next 256 bits form F (finalization key).
  3. The internal state Y is zeroed, and the nonce is stored.
- **Mathematical Meaning:**
  H and F are used as random multipliers in the universal hash construction that underlies QMAC.

6.2 Update: `qsc_qmac_update`

- **Input:** A message block X (32 bytes).
- **Operation:**
  1. X is XORed into the current state Y:

$$Y \leftarrow Y \oplus X.$$

  2. The updated state is then multiplied by H via the GF(2^256) multiplication:

$$Y \leftarrow \text{gfmul}(H, Y).$$

- **Mathematical Meaning:**
  Each message block is absorbed into Y and then mixed by the multiplication with H, ensuring that every bit of X affects the final MAC output.

6.3 Finalize: `qsc_qmac_finalize`

- **Input:** The final state Y (after all blocks are absorbed).
- **Operation:**
  1. The finalization key F is XORed with Y:

$$Y \leftarrow Y \oplus F.$$

  2. The resulting state is output as the MAC tag.
  3. If a nonce is maintained, it is incremented for future invocations.
- **Mathematical Meaning:**
  Finalization applies an additional random mask (from F) to the state, ensuring that any structural properties of the hash do not reveal information about the message or key.

6.4 Dispose: `qsc_qmac_dispose`

- **Operation:**
  The sensitive state (subkeys H, F and state Y) is cleared (zeroed) from memory, and the initialized flag is reset.
- **Mathematical Meaning:**
  Securely erasing the state prevents residual information from being exploited in side-channel or subsequent attacks.

7. Cryptanalytic Analysis and Security Estimates

7.1 Diffusion and Linear Mixing

The cmul256 operation, via Karatsuba's algorithm, ensures that every input bit is propagated into many output bits of the 512-bit product. The subsequent reduction $r = L \oplus [H \oplus (H \ll 19)]$ further diffuses the contributions of the upper half H across the final 256-bit result. Compared to GF(2^128) multiplication used in GMAC, the 256-bit multiplication has more degrees of freedom and inherently greater diffusion.

7.2 Security Estimates

- **Using SHAKE-256 for Key Derivation:**
  - *Classical:* The derived subkeys H and F are assumed to have 256-bit security, yielding an effective MAC collision resistance of approximately 2^{128} (by birthday bound).
  - *Quantum:* Grover's algorithm reduces the effective security to about 128 bits, permitting around 2^{64} quantum queries.
- **Using SHAKE-512 for Key Derivation:**
  - *Classical:* With SHAKE-512, the subkeys have a 512-bit security bond; the MAC's collision resistance becomes nearly 2^{256}.
  - *Quantum:* The effective security under Grover's algorithm is roughly 256 bits, meaning about 2^{128} quantum queries would be required.

7.3 Comparison with GMAC

GMAC (over GF(2^128)) typically provides a 128-bit tag with a classical security bound of approximately 2^{64} queries and a quantum bound of about 2^{32} queries. In contrast, QMAC's 256-bit output allows:

- Approximately 2^{128} classical queries when using SHAKE-256 (and up to 2^{256} when using SHAKE-512, before the birthday bound takes effect),
- And roughly 2^{128} quantum queries, depending on the XOF used for key derivation.

Thus, QMAC provides significantly improved bounds in both the classical and quantum settings.

8. Conclusion

QMAC is a robust MAC function that extends the GMAC construction to a 256-bit field. Its core operations are:

- A 256-bit carry-less multiplication (cmul256) computed via a Karatsuba algorithm using 128-bit cmul primitives.
- A reduction modulo $m(x) = x^{256} + x^{19} + 1$ given by:

$$r = L \oplus (H \oplus (H \ll 19)),$$

  where L and H are the lower and upper 256-bit halves of the 512-bit product.

- Subkey derivation using a SHA3/SHAKE-based construction (with either SHAKE-256 or SHAKE-512), offering quantum-resistant keying material.

The AVX/AVX2 implementations leverage __m256i registers for high throughput, while the AVX version using __m128i registers offers portability. The cryptanalytic analysis shows that the diffusion properties of the cmul256 operation are strong, and the overall construction provides a much larger security margin than GMAC. In particular, when using SHAKE-512, the effective quantum security can reach approximately 256 bits, thereby significantly reducing the adversary's ability to perform quantum queries.

QMAC's design, grounded in well-established GF(2) arithmetic and strengthened by the use of SHAKE-based subkey derivation, makes it a strong candidate for post-quantum applications requiring a high-security MAC.