AES-GCM-SIV: Specification and Analysis

Shay Gueron[1], Adam Langley[2], and Yehuda Lindell[3]*

[1] University of Haifa, Israel and Amazon Web Services
[2] Google, Inc.
[3] Bar-Ilan University, Israel

July 16, 2017

Abstract. In this paper, we describe and analyze the security of the AES-GCM-SIV mode of operation, as defined in the CFRG specification [10]. This mode differs from the original GCM-SIV mode that was designed in [11] in two main aspects. First, the CTR encryption uses a 127-bit pseudo-random counter instead of a 95-bit pseudo-random value concatenated with a 32-bit counter. This construction leads to improved security bounds when encrypting short messages. In addition, a new key derivation function is used for deriving a fresh set of keys for each nonce. This addition allows for encrypting up to 2^{50} messages with the same key, compared to the significant limitation of only 2^{32} messages that were allowed with GCM-SIV (which inherited this same limit from AES-GCM). As a result, the new construction is well suited for real world applications that need a nonce-misuse resistant Authenticated Encryption scheme. We explain the limitations of GCM-SIV, which motivate the new construction, prove the security properties of AES-GCM-SIV, and show how these properties support real usages. Implementations are publicly available in [8]. We remark that AES-GCM-SIV is already integrated into Google's BoringSSL library [1] and is deployed for ticket encryption in QUIC [17].

Preamble for the July 2017 edition

We would like to thank Tetsu Iwata and Yannick Seurin for alerting us to the fact that we had erroneously assumed that one of the terms in the security bounds of AES-GCM-SIV was dominated by another term. (Specifically, ϵ', the advantage of the adversary $\mathcal{A}'$; see comments on pages 11 and 12 of the previous version of this paper). Thus, while the security proof was correct, the example concrete bounds were overly optimistic, most notably for very large messages.

This July 2017 update fixes the concrete bounds that were given in the previous version, and some other small errors pointed out by Iwata & Seurin. Detailed proofs for the bounds will soon be released in [12].

* Suppored by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office.

1 Introduction

The concept of Authenticated Encryption with Additional Authenticated Data (AEAD) couples confidentiality and integrity in a mode of operation that should be easy for practitioners to use correctly. The most popular AEAD today, AES-GCM [13,14], is seeing widespread use due to its attractive performance, which is enhanced by AES and polynomial multiplication instructions that are now part of many modern processor architectures. However (like most AEADs), it suffers catastrophic failures of confidentiality and integrity if two distinct messages happen to be encrypted, under the same key, with the same nonce. While the requirements for AEADs specify that the pair of (key, nonce) shall only ever be used once, and thus prohibit such failures, there are cases where, in practice, guaranteed uniqueness of nonces is a concern. Nonce-misuse resistant AEADs [15] do not suffer from this problem. For this class of AEADs, encrypting two messages with the same nonce only discloses whether the messages were equal or not. This is the minimum amount of information that a deterministic algorithm can leak in this situation.

Gueron and Lindell [11] proposed efficient constructions for a nonce-misuse resistant AEAD (nmrAEAD) scheme, called GCM-SIV, that can use the same CPU instructions that accelerate AES-GCM. (Hereafter, GCM-SIV refers to the two-key variant of [11], with AES as the block cipher.) GCM-SIV takes an integer $k \leq 32$ as part of its context, which limits the message lengths to at most $2^k - 1$ blocks. The default value is $k = 32$ (i.e., messages can take the maximum length that GCM-SIV supports). GCM-SIV has different performance characteristics for encryption and decryption. For encryption, it is slower than AES-GCM, because achieving nonce-misuse resistance requires, by definition, two (*serialized*) passes over the data. Nevertheless, optimized implementations run GCM-SIV (for 128-bit keys) at less than one cycle per byte on modern processors (roughly 2/3 of the speed of nonce-respecting AES-GCM). On the other hand, GCM-SIV decryption runs at almost the same speed as AES-GCM.

GCM-SIV is formally described in [11] (a formal description can be derived from Fig. 1). Informally, the algorithm uses a hash key ($K1$) and an encryption key ($K2$), applies a universal hash function (GHASH) with $K1$ to the encoded AAD (additional authentication data) and MSG (plaintext to be encrypted), and generates an authentication tag by AES-encrypting the hash value, XOR-ed with the nonce, under $K2$. Finally, the plaintext MSG is encrypted with AES in CTR mode, using $K2$, and with an initial counter derived from the authentication tag. This strategy means that the initial counter (effective nonce for the CTR encryption) is pseudorandom for every different nonce/message pair. Thus, even if the actual nonce repeats, the effective nonce used to mask the encryption is different for different messages. The initial counter is of length $n - k$ bits, and the value that is incremented is of length k bits (this is like in AES-GCM where $k = 32$ is fixed). Thus, the maximum message length is $2^k - 1$ blocks. In the general constructions in [11], an arbitrary pseudorandom function F was referred to; here the specific instantiation is either AES128 or AES256, and this affects the length of the key $K2$. The security bounds of GCM-SIV were proven in [11, Theorem 4.3], as follows:

Theorem 1 (Theorem 4.3 of [11] (2-Key GCM-SIV)). *The GCM-SIV mode of operation is a nonce-misuse resistant authenticated encryption scheme. Furthermore, for any fixed parameter $k < n$ in GCM-SIV determining the maximum message length of $2^k - 1$, and for every adversary $\mathcal{A}$ attacking the GCM-SIV construction, making q_E encryption queries and q_D decryption queries, there exists an adversary $\mathcal{A}'$ for distinguishing F from a random function, such that*

$$\mathbf{Adv}_{\Pi}^{\mathrm{mrAE}}(\mathcal{A}) < 2 \cdot \mathbf{Adv}_{F}^{\mathrm{prf}}(\mathcal{A}') + \frac{\left(\lceil \frac{\bar{M}}{n} \rceil + 1\right) \cdot (q_E(\mathcal{A}) + q_D(\mathcal{A}))^2}{2^{n-1}} + \frac{q_E(\mathcal{A})^2}{2^{n-k-2}} \qquad (1)$$

where $t(\mathcal{A}') \leq 6 \cdot t(\mathcal{A})$ and $q_f(\mathcal{A}') \leq 2q_E(\mathcal{A}) + 2q_d(\mathcal{A}) + \frac{L}{n}$, the value L is the overall length of all encrypted or decrypted messages, and $\bar{M}$ is an upper bound on the length of all encryption and decryption queries including the length of the message plus the AAD[1]*.*

[1] For simplicity, we ignore an additive term $q_D(\mathcal{A})/2^n$, which is dominated by (implied) $q_D(\mathcal{A})^2/2^n$. We also assume that AES satisfies its design goals, and implicitly ignore PRP advantage of AES.

In Theorem 1, $t(\mathcal{A}')$ and $t(\mathcal{A})$ denote the running times for adversaries $\mathcal{A}$ and $\mathcal{A}'$, respectively, and q_f is the number of oracle queries to the pseudorandom function oracle (for adversaries distinguishing a pseudorandom function from a random one). Recall that in the specific GCM-SIV instantiation, the pseudorandom function F is AES (with key $K2$).

Safety margins for using GCM-SIV, and the implied limit on the lifetime of a key. A loose interpretation of Theorem 1 is that an adversary against GCM-SIV, who uses a given budget of encryption/decryption queries, has advantage that is at most twice the advantage that any adversary has, for distinguishing AES outputs from random (using roughly the same time and queries budget), plus the two last terms in the RHS of (1). The term $q_E(\mathcal{A})^2/2^{n-k-2}$ represents the probability that "randomized" IV's, which are used for the encryption, would collide. If we assume that a usage requires support for messages with the maximal allowed length of $2^{32}-1$ blocks ($k=32$), this term is $q_E(\mathcal{A})^2/2^{94}$. Note also that this bound in (1) is essentially tight: after encrypting $\sim 2^{47}$ messages, there is a high probability of a collision in the 95-bit pseudorandom value $TAG[126:k]$ (recall that $k=32$) that initializes the CTR block used in the encryption step (see Step 16 in Fig. 1). Since such a collision results in the use of the same stream for encryption, we must place a limit on the number of GCM-SIV encryptions that provides safe security margins. This limit should handle the collision probability as a minimum (note that there are larger terms in the RHS of (1)).

In order provide a recommendation on the maximal number of GCM-SIV encryptions (with the same key), it is useful to refer to NIST's guidelines [5] for AES-GCM with a random 96-bit IV (or any IV whose bitlength is not 96), which faces an analogous situation. The NIST requirement is that the probability of an IV collision should not exceed 2^{-32}, and this is translated in [5] to limiting the allowed number of encryptions with AES-GCM using a random IV to 2^{32}. With the same rationale, a limit of 2^{31} GCM-SIV encryptions (with the same key) is appropriate, as a minimum, since $(2^{31})^2/2^{94}=2^{-32}$. On top of this, we wish to require that all of the terms in the security bounds that appear in (1) will be bounded by $\sim 2^{-32}$, and this imposes even stricter constraints in the maximal allowed number of encryptions. Unfortunately, the implied restriction on the lifetime of a single key is problematic for some real usages that wish to deploy GCM-SIV (e.g., when relatively short messages are encrypted at very high frequency). AES-GCM-SIV is designed to address this problem.

AES-GCM-SIV – the CFRG proposal. Due to the aforementioned limitation on the allowed number of encryptions using GCM-SIV, the concrete CFRG proposal [10] (which we call AES-GCM-SIV) differs in two important ways:

1. *Key derivation:* The most important difference is that encryption begins by deriving keys from a master key and the nonce. This provides the property that "independent" keys are used for different nonces. This derivation must be carried out carefully in order to obtain good security bounds. In particular, a careless method will itself suffer from birthday bounds on collisions and so may not yield very good bounds. In addition, the key derivation method must also be efficient. We set our goal to achieve a simple and efficient key derivation mechanism, where the derived keys can be distinguished from random with advantage of at most 2^{-32}, even after $\sim 2^{64}$ derivations.
2. *Counter generation:* As mentioned above, GCM-SIV essentially generates a 95-bit pseudorandom value that initializes the counter block in the encryption phase. Similarly to the SIV mode of operation [15], we initialize the counter block to be (almost) a full random block. This provides much better collision bounds when many short messages are encrypted.

In addition to the above, AES-GCM-SIV differs from GCM-SIV in the exact specification of the universal hash function used in the tag generation. GCM-SIV uses the GHASH function of AES-GCM; in contrast, AES-GCM-SIV uses a hash function that we call POLYVAL, which is very similar to GHASH but avoids the byte swapping which slows down implementations. Our optimized implementation of POLYVAL is 1.2 times faster than the GHASH implementation in OpenSSL 1.0.2k.

As is formally shown in [12], the dominating term in the security bound for AES-GCM-SIV is reduced from $\frac{q_E(\mathcal{A})^2}{2^{n-m-2}}$ to $\max\left\{\frac{Q\cdot B\mathsf{max}^2}{2^{n+1}}, \frac{\sum_{i=1}^{Q}(N_E^i)^2}{2^{n-m-2}}\right\}$, where Q is the number of different nonces used (in encryption or decryption queries), $B\mathsf{max}$ is the maximum number of blocks encrypted with a single nonce, and N_E^i is the number of messages encrypted with the ith nonce. As can clearly be seen, in the typical case where nonces repeat infrequently, this reduces the bound from a term that is *quadratic* in the number of encryptions to a term that is (almost) *linear* in the number of encryptions. Thus, for a large number of encryptions, a much better security margin is achieved. We have:

Theorem 2 (Corollary 6.3 of [12] – informal). *Assume the standard bounds regarding the indistinguishability of AES from a random permutation and a random function. Then, for every nonce adversary $\mathcal{A}$ attacking AES-GCM-SIV using $Q \leq 2^{64}$ different nonces in both encryption and decryption queries, querying the ith nonce with N_E^i different messages, and querying any single nonce with at most $B\mathsf{max}$ blocks overall and maximum message-length 2^m for any single message, we have that $\mathcal{A}$'s advantage in the nonce-misuse resistance experiment is at most*

$$\frac{3Q}{2^{96}} + \frac{Q \cdot B\mathsf{max}^2}{2^{129}} + \frac{\sum_{i=1}^{Q}(N_E^i)^2}{2^{126-m}}.$$

We remark that this is an informal description of the bound, and it also assumes that the overall length of all decryption queries is less than $Q \cdot B\mathsf{max}^2$; this holds in typical applications (the possible exception is where the adversary can issue decryption queries with extremely long AAD, and the system would agree to process them).

An example use case – QUIC. AES-GCM-SIV can be used anywhere where nonce-misuse resistance is desired, with security bounds that are never lower than GCM-SIV, and are significantly higher in some cases. For one example of such a use case, we consider a scenario where a large number of (not necessarily short) messages are encrypted by a single "user", and it is difficult to ensure that nonces are always unique. We remark that such a "user" can be an application that runs over multiple servers. A real world example of such a scenario is QUIC.

QUIC [17] is a new transport protocol that is designed to improve the perceived performance of connection-oriented web applications that are currently using TCP, while providing security protection that is comparable to that of TLS. QUIC needs to encrypt "source-address tokens", such that a cluster of servers can recognize them in the future, but without clients being able to forge them. Simply adding a MAC would suffice, but for future-proofing they should also be confidential. All servers can share a fairly long-lived secret key, but the servers need to be able to create these tokens quickly, and independently. Since a central allocation system for nonces is not operationally viable, random selection of nonces is the only possibility. AES-GCM's limit of 2^{32} random nonces (per key) suggests that, even if the system rotated these secret keys *daily*, it could not issue more than about 50K tokens per second. However, in order to process DDoS attacks the system may need to sustain issuance of several hundred million per second. A similar problem arises in TLS with session tickets [2]. Although the demands are significantly reduced in this context, a limit of 50K tickets per second is still insufficient for many sites, and thus plain AES-GCM is unsuitable for this as well. AES-GCM-SIV is a possible solution. Indeed, AES-GCM-SIV is already integrated into BoringSSL [1] (Google's fork of OpenSSL), and Google is planning to use AES-GCM-SIV for QUIC's source-address tokens.

Organization. In Section 2, we present general preliminaires, as well as a description of the POLYVAL universal hash function. Then, in Section 3, we present an intermediate mode of operation, denoted GCM-SIV$^+$, that differs from GCM-SIV by generating the counter as an almost full random block and used POLYVAL instead of GHASH. We analyze the security bounds of this scheme as part of our modular analysis of AES-GCM-SIV. Following this, in Section 4 we present our key derivation function and analyze its properties. The full AES-GCM-SIV mode of operation is then derived in Section 5 by incorporating the key derivation function into GCM-SIV$^+$. Finally, in Section 6 we provide performance results and comparisons, and in Section 7 we discuss the bounds obtained in different usage scenarios.

2 Preliminaries and Notation

2.1 General Notation

In this section, we present some general notation; this notation appears in [10] and is reproduced here for the sake of clarity and completeness.

Let $n > 0$ be an integer, and let $\{0,1\}^n$ be the set of all n-bit strings. An element in $A \in \{0,1\}^n$ is identified as an n-bit string $A = a_{n-1}a_{n-2}\dots a_0$, where, for $0 \le j \le n-1$, a_j is the bit in position j of A. By convention, we write here the strings with the most significant bit (a_{n-1}) in the leftmost position, and the least significant bit (a_0) in the rightmost position. We view $\{0,1\}^n$, interchangeably, as the set of integers between 0 and $2^n - 1$, where the string A corresponds to the binary representation of the integer $\tilde{A} = \sum_{j=0}^{n-1} a_j 2^j$. With the above writing convention, $a_0 = \tilde{A} \pmod 2$.

Let $0 \le i < 2^{32}$ be an integer, and let str be a 32-bit string. Then, $\mathsf{IntToString32}(i)$ is the 32-bit string that encodes the binary representation of i, and $\mathsf{StringToInt32}(str)$ is the integer x whose 32-bit binary representation is str. For example, $\mathsf{IntToString32}(2^{30} - 5) = 00111111111111111111111111111011$ and $\mathsf{StringToInt32}(00000000000000000000000000001011) = 11$. For 64 bits and integers $0 \le i < 2^{64}$, we use $\mathsf{IntToString64}$ and $\mathsf{StringToInt64}$ analogously. Repeating bits can also denoted with a superscript that counts the repetition, for example 0^{127} is the string of 127 zero bits.

Let $X = X[\ell - 1 : 0]$ be a string of ℓ bits, and let $k1, k2$ be such that $0 \le k1 < k2 \le \ell - 1$. Then, $\mathsf{bitlen}(X) = \ell$, and if ℓ is divisible by 8 (i.e., X is a string of bytes), $\mathsf{bytelen}(X) = \ell/8$. The notation $X[k2 : k1]$ represent the sub string of X, with $k2 - k1 + 1$ bits, from the appropriate positions. Concatenation of strings is denoted by $\|$. For example, if $\ell = 16$ and X is the string that holds the binary representation of the integer 37449, then $X[15 : 0]$=1001001001001001, $\mathsf{bitlen}(X) = 16$, $\mathsf{bytelen}(X) = 2$, $X[12 : 8] = 10010$.

A "block" is a string of 128 bits (16 bytes). A block X is denoted by its bits as $X[127 : 0]$, and can also be denoted as a string of 16 bytes $B_{15}B_{14}\dots B_0$. By convention, we write the most significant byte (B_{15}) in the leftmost position, and the least significant bit (B_0) in the rightmost position. We use AES128(K, X) / AES256(K, X) to denote the AES encryption of a block X, using the key K of 128/256 bits, respectively. For example, if ZERO is the block of 128 zero bits, then AES128 (ZERO, ZERO) is written, as a 16-byte string under our convention, as 2e2b34ca59fa4c883b2c8aefd44be966. Here, the least significant byte of A (0x66) is in the rightmost position. Bits number $0, 64, 127$ are, respectively, $a_0 = 0$, $a_{64} = 0$, $a_{127} = 0$.

We define the operation $\mathsf{ByteSwap}$ as follows. If $X = B_{15}B_{14}\dots B_0$ is a block, then $\mathsf{ByteSwap}(X) = B_0B_1\dots B_{14}B_{15}$, i.e., the block which has the same bytes as X, but written in the reverse order. For example,

$$\mathsf{ByteSwap}\ (\text{2e2b34ca59fa4c883b2c8aefd44be966}) = \text{66e94bd4ef8a2c3b884cfa59ca342b2e}.$$

According to the relevant context, we view, interchangeably, elements in the set $\{0,1\}^n$, as elements in the finite field $GF[2^n]$ with 2^n elements. These elements are formal binary polynomials of degree $n-1$, and thus the string $A = a_{n-1}a_{n-2}\dots a_0 \in \{0,1\}^n$ corresponds to the polynomial $A = A(x) = \sum_{j=0}^{n-1} a_j x^j$. A specific representation of the field is defined by choosing an irreducible reduction polynomial, $Q(x)$, of degree exactly n, and defining the field operations $+$ and $\otimes$ through polynomial arithmetic modulo $Q(x)$. Such a representation is denoted by $\mathbb{F}_{2^n}[x] \Big/ Q(x)$. Here, field addition $+$ is polynomial addition in the ring $\mathbb{Z}_2[x]$ (which corresponds to a bitwise XOR of the corresponding bit strings). Field multiplication $A(x) \otimes B(x)$ is the polynomial $D(x) = A(x) \times B(x) \pmod{Q(x)}$, where $\times$ denotes polynomial multiplication in the ring $\mathbb{Z}_2[x]$, and the polynomial division modulo $Q(x)$ is carried out over $\mathbb{Z}_2[x]$. The multiplicative unit and the zero element in $\mathbb{F}$ are the 128-bit strings $0^{127}\|1$ and 0^{128}, respectively.

2.2 The Polynomial Evaluation Hash Function POLYVAL

AES-GCM-SIV uses a polynomial evaluation hash function called POLYVAL, rather than the GHASH function, used by AES-GCM. We define this function here, and compare it to GHASH in Section 2.3.

We choose a representation of the finite field with 2^{128} elements via the reduction polynomial $Q(x) \equiv x^{128} + x^{127} + x^{126} + x^{121} + 1$ (i.e., $GF(2^{128})[x] \Big/ \left(x^{128} + x^{127} + x^{126} + x^{121} + 1\right)$). For short, we denote the field in this representation by $\mathbb{F}$. Let "$\bullet$" be the following operation in $\mathbb{F}$. For every $A_1, A_1 \in \mathbb{F}$,

$$A_1 \bullet A_2 = A_1 \otimes A_2 \otimes x^{-128}. \tag{2}$$

Definition 1 **($\mathbb{F}^*$)** *We denote the set $\mathbb{F}^* = \{(X_1, \ldots, X_v) \mid v \geq 1, X_1, \ldots, X_v \in \mathbb{F}\}$ of all vectors of length at least 1 of arbitrary elements in $\mathbb{F}$. A vector in $\mathbb{F}^*$ is called a "message". For a specific message $X_1, \ldots, X_u$, we say that the length of the message is u.*

Definition 2 (The POLYVAL Hash Function) *Define $POLYVAL : \mathbb{F} \times \mathbb{F}^* \to \mathbb{F}$ to be a keyed function with key-space $\mathbb{F}$, message space $\mathbb{F}^*$ and range $\mathbb{F}$. We denote the key by $H \in \mathbb{F}$, and define $H_j = H^j \otimes x^{-128(j-1)}$, for $j = 1, 2, \ldots, s$ and s any positive integer. Then, the polynomial evaluation hash of the message $X_1, X_2, \ldots, X_s$, using the hash key H, is defined by*

$$POLYVAL(H, X_1, X_2, \ldots, X_s) = X_1 \bullet H_s + X_2 \bullet H_{s-1} + \ldots + X_{s-1} \bullet H_2 + X_s \bullet H_1. \tag{3}$$

The POLYVAL family of hash functions is defined to be the set $\{POLYVAL(H, \cdot)\}_{H \in \mathbb{F}}$.

We remark that using the definition in (2), the POLYVAL definition in (3) is equivalent to

$$POLYVAL(H, X_1, X_2, \ldots, X_s) = \sum_{i=1}^{s} \left(X_i \otimes H^{s-i+1} \otimes x^{-128 \cdot (s-i+1)} \right) \tag{4}$$

Iterative computation of POLYVAL: In order to compute POLYVAL efficiently, first observe that H_j can be computed iteratively by setting $H_1 = H$ and then $H_j = H_{j-1} \bullet H$, for $j = 2, \ldots, s$. Likewise, POLYVAL can be computed iteratively using Horner's method by setting $S_0 = 0$ and $S_j = (S_{j-1} + X_j) \bullet H$, for $j = 1, \ldots, s$. Finally, the output is $POLYVAL(H, X) = S_s$.

Security. The GCM-SIV construction, that we build upon, uses an ϵ-XOR universal family of hash functions. We define this notion now and then prove that POLYVAL achieves the definition.

Definition 3 (ϵ-XOR universal family of hash functions) *Let $\mathcal{H} = \{h_H : \mathbb{F}^* \to \mathbb{F}\}_{H \in \mathbb{F}}$ be a family of hash functions that are indexed by a hash key $H \in \mathbb{F}$. $\mathcal{H}$ is an ϵ-XOR universal family of hash functions if, for any two distinct inputs $X, Y \in \mathbb{F}^*$, and any element $A \in \mathbb{F}$, it holds that*

$$\Pr_{H \in \mathbb{F}}[h_H(X) \oplus h_H(Y) = A] \leq \epsilon(n).$$

Lemma 3. *POLYVAL is an ϵ-XOR universal family of hash functions with $\epsilon = smax/2^{128}$, where $smax$ is an upper bound on the length of the input messages.*

Proof. Let $H \in \mathbb{F}$ be a hash key, and let $A \in \mathbb{F}$ be an element. Let $X_1, X_2, \ldots, X_{s_1}$ and $Y_1, Y_2, \ldots, Y_{s_2}$ be two *distinct* messages of lengths s_1 and s_2, respectively, such that $smax \geq s_1 \geq s_2 \geq 1$. By prepending zero elements to the shorter message, we may assume that $s_1 = s_2 = s$ (since zero elements do not change the result), and denote $Z_j = X_j + Y_j$, $j = 1, \ldots, s$. Note that at least one of the Z_j values (not necessarily Z_s) is nonzero, since $X \neq Y$. Given the above, the equality

$$POLYVAL(H, X_1, X_2, \ldots, X_s) + POLYVAL(H, Y_1, Y_2, \ldots, Y_s) = A$$

is equivalent to

$$POLYVAL(H, Z_1, Z_2, \ldots, Z_s) = A. \tag{5}$$

Denoting $W = H \otimes x^{-128}$, and using the equivalent definition in Eq. (4), we conclude that the equality in Eq. (5) is equivalent to

$$Z_1 \otimes W^s + \ldots Z_s \otimes W - A = 0. \tag{6}$$

Equation (6) implies that W is a root of a non-zero polynomial of degree at most s over the field $\mathbb{F}$. Now, every $H \in \mathbb{F}$ defines a unique W, and thus when H is uniformly distributed we have that W is uniformly distributed. Since there are at most s roots of the polynomial in Eq. (6), we have that the probability that a uniform W is a root is at most $\frac{s}{2^{128}}$. Since $s \leq smax$, we obtain the desired bound of $\frac{smax}{2^{128}}$. □

We remark that Lemma 3 holds as long as all messages have unique encodings. That is, denote by Encode(`AAD`, `MSG`) the encoding of the additional authenticated data `AAD` and plaintext message `MSG`, as described in Lines 10–11 of Figure 1. Then, since the last block contains the length of `AAD` concatenated with the length of `MSG` (in a unique way), we have that for every (`AAD`, `MSG`), (`AAD'`, `MSG'`) with (`AAD`, `MSG`) $\neq$ (`AAD'`, `MSG'`), it holds that Encode(`AAD`, `MSG`) $\neq$ Encode(`AAD'`, `MSG'`).

2.3 Choosing POLYVAL Over GHASH

In this section we explain the reason for using the POLYVAL hash function in AES-GCM-SIV, rather than the GHASH function which is used in AES-GCM. Indeed, both leverage the same properties of polynomial hashing and, as we show now, are very closely related. The following lemma from [9], shows the relation between POLYVAL and GHASH.

Lemma 4. *The POLYVAL and GHASH functions fulfill the following identity*

$$\begin{aligned}&POLYVAL(H, X_1, X_2, \ldots, X_s)\\&\quad = \mathsf{ByteSwap}\left(GHASH\left(\mathsf{ByteSwap}(H \otimes x), \mathsf{ByteSwap}(X_1), \mathsf{ByteSwap}(X_2), \ldots, \mathsf{ByteSwap}(X_s)\right)\right).\end{aligned}$$

Lemma 4 shows that it is possible to compute POLYVAL by using a given function that computes GHASH. This can be done by calling that function with a byte-swapped hash key multiplied by $x \in \mathbb{F}$ and byte-swapped input blocks, and then byte swapping the result. Such an implementation can be useful in cases where an implementer wishes to re-use software/hardware in an existing AES-GCM implementation. However, this requires additional byte swapping which comes at a cost.

We now show that a *direct* implementation of POLYVAL is preferable to a *direct* implementation of GHASH on x86-64 architectures (with AES-NI and PCLMULQDQ instructions). On such architectures, the computations of GHASH require a transformation of the hash key, plus a byte swap of every message block. By contrast, direct computations of POLYVAL, carried out in $\mathbb{F}$ as described above save the need to for byte swapping and transforming the hash key. In this light, the identity in (7) is expressing a *saving* of unnecessary operations that GHASH adds.

Experimental results. We implemented POLYVAL and compared its running time to the highly optimized implementation of GHASH in OpenSSL 1.0.2k. (The results appear in Figure 4 in Section 6.) Asymptotically, POLYVAL is 1.2 times faster than GHASH and is faster for all byte lengths.

3 The Intermediate GCM-SIV$^+$ Mode of Operation

In this section, we describe a variant of the original GCM-SIV [11], denoted GCM-SIV$^+$. This variant is presented separately for the sake of providing a modular analysis of AES-GCM-SIV. GCM-SIV$^+$differs from GCM-SIV in two ways; first, the POLYVAL hash function is used instead of the GHASH function, and the initial block is not defined to be of length $128 - m - 1$ bits (where $2^m - 1$ is the maximum number of message blocks), but rather to be of length $n - 1$ bits. This algorithm is described in Fig. 1, with its differences to GCM-SIV.
We now prove that the bound of Theorem 1 for GCM-SIV applies also to GCM-SIV$^+$, but with one important difference. In Theorem 1, the value k is the fixed parameter determining the length of the tag in GCM-SIV. In contrast, in the following theorem, the value m (that replaces k) is the maximum-length of any message encrypted by the adversary. Thus, the *order of quantifiers* regarding k and m in the following theorem is different to that of Theorem 1. This makes a big difference if many small messages are encrypted. We remark that this way of determining the counter is the same as in the SIV construction of [16], but differs from GCM-SIV in [11].

Theorem 5 (Security bounds of GCM-SIV$^+$.). *The GCM-SIV$^+$mode of operation is a nonce-misuse resistant authenticated encryption. Furthermore, for every adversary $\mathcal{A}$ attacking GCM-SIV$^+$, making q_E encryption queries of maximum length $2^m - 1$ with $m \leq 32$, and q_D decryption queries, there exists an adversary $\mathcal{A}'$ for distinguishing F from a random function, such that*

$$\mathbf{Adv}_{\Pi}^{\mathrm{mrAE}}(\mathcal{A}) < 2 \cdot \mathbf{Adv}_{F}^{\mathrm{prf}}(\mathcal{A}') + \frac{q_E(\mathcal{A})^2}{2^{n-m-2}} + \frac{(M+1)\cdot(q_E(\mathcal{A}) + q_D(\mathcal{A}))^2}{2^{n-1}} \tag{7}$$

where $t(\mathcal{A}') \leq 6 \cdot t(\mathcal{A})$ and $q_f(\mathcal{A}') \leq 2q_E(\mathcal{A}) + 2q_D(\mathcal{A}) + \frac{L}{n}$, the value L is the overall length of all encrypted or decrypted messages, and M is an upper bound, over all encryption and decryption queries, on the number of blocks in a message plus the AAD[2].

[2] Again, ignoring a dominated additive term $q_D(\mathcal{A})/2^n$.

GCM-SIV$^+$ (encryption-keylength, K1, K2, N, AAD, MSG)

```
1.   Context: encryption-keylength (= 128 or 256)
              0 <=  m <= 32 such that MSG length is at most 2^m-1 blocks.
2.   Keys: K1 (128 bits), K2 (128 or 256 bits)
3.   If encryption-keylength = 128, AES = AES128, else AES = AES256
4.   Input: AAD, MSG, N (96 bits)
5.   Padding:
6.     A = Zero pad AAD to the next 16 bytes boundary (d blocks)
7.     M = Zero pad MSG to the next 16 bytes boundary (v blocks)
8.       (denote M by blocks as: M0, M1, ..., M(v-1).)
9.   Encrypting and Authenticating:
10.    L1 = (bytelen(AAD)*8); L2 = (bytelen(MSG)*8)
11.    LENBLK = IntToString64(L1) || IntToString64(L2)
12*.   T = POLYVAL (K1, A || M || LENBLK)
13.    TAG = AES (K2, 0 || (T XOR N) [126:0])
14.    for i = 0, 1, ..., v-1 do
15*.     Low32(i)  = (StringToInt32(TAG[31:0]) + i) mod 2^{32}
16*.     CTRBLK_i = 1 || TAG[126:32] || IntToString32(Low32(i))
17.      CTi = AES (K2, CTRBLK_i) XOR Mi
18.    end do
19.    Set C = CT0, CT1, ..., CT(v-1)
20.    if length(MSG) != length(CT)
21.      Chop off lsbytes of CT(v-1) to make lengths equal
22.  Output: C = (CT0, CT1, ..., CT(v-1)), TAG

------------GCM-SIV-------------
12*.    GCM-SIV used GHASH instead of POLYVAL
15-16*. GCM-SIV set CTRBLK_i = 1 || TAG[126:k] || IntToString32(i)
--------------------------------
```

Fig. 1. Specification of GCM-SIV$^+$. The differences between GCM-SIV$^+$and GCM-SIV are in Steps 12*, 15* and 16*.

Proof. The statement follows from the proof of Theorem 4.3 in [11], using the fact that POLYVAL is a universal hash function (Lemma 3; this relies on the prefix-free encoding due to Lines 10–11 in Figure 1, as mentioned after the proof of Lemma 3) and the fact that the $\frac{q_E(\mathcal{A})^2}{2^{n-m-2}}$ factor is due to the collision probability in $TAG[126:m]$ in GCM-SIV. Thus, we conclude the proof of the proposition by showing that with the counter method of GCM-SIV$^+$, the collision probability is $\frac{q_E(\mathcal{A})^2}{2^{n-m-2}}$ where $2^m - 1$ is the maximum message length. This follows from a standard birthday analysis, based on the length of TAG being $n-m-1$. $\square$

4 DeriveKey – Efficient Key Derivation With Good Bounds

We describe a new Key Derivation Function (KDF) here, that is both efficient to implement and obtains very good bounds. Recall that our goal is to obtain a simple and efficient design, where the indistinguishability (from random) of the derived keys is at most to 2^{-32} even after $\sim 2^{64}$ derivations.

The KDF works by truncating outputs of a pseudorandom permutation. Concretely, we apply the AES pseudorandom permutation to the input nonce and an index, and truncate each 128-bit output to 64 bits. Thus, a 128-bit key is derived by applying AES twice, and a 256-bit key is derived by applying AES four times. In GCM-SIV$^+$, we require an AES key of length 128 or 256, and a POLYVAL hash key of length 128. Thus, the KDF involves 4 AES invocations for AES128 and 6 AES invocations for AES256. We use a nonce of 96-bits, since this is standard practice for existing AES-GCM interfaces. The algorithm is described in Fig. 2.

Intuitively, truncating the output of AES is advantageous since it lowers the distinguishing probability of AES from a pseudorandom *function*. Specifically, using a random permutation has the disadvantage that

DeriveKey(K, N)

```
Context: encryption-keylength (= 128 or 256)
if encryption-keylength = 128 AES is AES128, else AES is AES256
Key: K
Input: N (96 bits)
If encryption-keylength =128 then repeats = 4, else repeats = 6
for i from 0 to repeats-1 do
  Tj = AES (K, N [95:0] || IntToString32 (i))
end
K1 = T1 [63:0] || T0 [63:0]
If keylength=128 then
   K2 = T3 [63:0] || T2 [63:0]
else
   K2 = T5 [63:0] || T4 [63:0] || T3 [63:0] || T2 [63:0]
end
Output: K1 (128 bits), K2 (128 or 256 bits)
```

Fig. 2. DeriveKey uses the KDF key K to derive two new keys: K1 (128 bits) and K2 (128 or 256 bits).

derived keys are distinguishable from random at around the birthday bound. In contrast, a random function suffers from no such limitation, and thus a pseudorandom function (versus permutation) is advantageous in this sense.

The following lemma is proven in [6], which explores the problem of distinguishing the truncation of a randomly chosen permutation from a random function. The upper bound on the distinguishing advantage (originally due to [18]), is simplified in [6] to the easy-to-use form

$$\mathbf{Adv}_{n,m}(\tilde{q}) \leq \min\left(\frac{\tilde{q}^2}{2^{n+1}}, \frac{\tilde{q}}{2^{\frac{m+n}{2}}}, 1\right) \tag{8}$$

where $\tilde{q} \leq \frac{3}{4} \cdot 2^n$ is the number of queries, and where the (randomly chosen) the permutation over n bits is truncated to $n - m$ bits (for some $1 \leq m < n$). We comment that Gilboa and Gueron [7] have recently proved that this bound is essentially tight. Plugging in $n = 128$ and $m = 64$ as in our DeriveKey procedure, we have:

Lemma 6 (The DeriveKey advantage). *Let $\mathcal{A}$ be an adversary that makes at most $Q \leq \frac{3}{24} \cdot 2^{128}$ queries to DeriveKey (obtaining Q pairs of keys, K_1 (128 bits), K_2 (128 or 256 bits)). Then,*

$$\mathbf{Adv}^{\mathsf{prf}}_{DeriveKey}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{prp}}_{AES}(\mathcal{A}') + \min\left\{\frac{36Q^2}{2^{129}}, \frac{6Q}{2^{96}}, 1\right\} \tag{9}$$

where adversary $\mathcal{A}'$ makes at most $6Q$ AES oracle queries.

Proof. To obtain one pair of keys (K_1 and K_2), DeriveKey computes at most 6 AES operations using the KDF key (for the 128-bit case, only 4 AES operations are required). Suppose an adversary $\mathcal{A}$ makes at most $Q \leq \frac{3}{24} 2^{128}$ queries to DeriveKey, and obtains Q pairs of keys K_1, K_2. Then, within Eq. (8), we can set $\tilde{q} = 6Q$, $n = 128$ and $m = 64$, and derive that

$$\frac{\tilde{q}^2}{2^{n+1}} = \frac{36Q^2}{2^{129}} \quad \text{and} \quad \frac{\tilde{q}}{2^{\frac{m+n}{2}}} = \frac{6Q}{2^{96}}$$

and the bound on the number of queries $\tilde{q} \leq \frac{3}{4} \cdot 2^n$ is equivalent to $6Q \leq \frac{3}{4} \cdot 2^n$, implying that $Q \leq \frac{3}{24} \cdot 2^n$. Finally, since AES is a pseudorandom permutation and not a truly random permutation, the bound includes $\mathbf{Adv}^{\mathsf{prp}}_{AES}(\mathcal{A}')$, which is $\mathcal{A}'$'s advantage in distinguishing AES from a random *permutation*. This completes the proof. ☐

Actual DeriveKey bounds. The crucial point to observe in Eq. (9) is that the advantage is the *minimum* of $O(Q/2^{96})$ and $O(Q^2/2^{192})$. Thus, the birthday bound of $Q \approx 2^{64}$ for distinguishing AES from a pseudorandom function does not arise here. Rather, at $Q = 2^{64}$, the distinguishing advantage is only $\frac{6}{2^{32}}$ since the linear term of $\frac{6Q}{2^{96}}$ is much smaller for large Q. Thus, it is possible to derive far more keys than by using counter-mode[3]. It is worth noting that for small values of Q, the quadratic term is smaller; however, the minimum is so small in these cases this is irrelevant. In conclusion, using the NIST bounds for AES-GCM that allow for 2^{-32} advantage, we are still able to derive approximately 2^{64} different keys.

Observe that $\frac{36Q^2}{2^{129}} < \frac{6Q}{2^{96}}$ if and only if $Q < 2^{33}/6$. Since we are interested in large values of Q, and since the bounds are so small for $Q < 2^{33}/6$, we use the bound $\frac{6Q}{2^{96}}$ from here on.

Efficiency. In order to compute DeriveKey, the number of AES invocations is 4 with encryption-keylength = 128, and 6 with encryption-keylength = 256. Importantly, these AES computations are parallelizable. Furthermore, the AES key schedule for the master key K can be pre-computed and cached, and so we can ignore the key expansion overhead. Thus, on a modern CPU with AES instructions (AES-NI) with throughput of 1 cycle and latency of 4 cycles, DeriveKey consumes ~ 50 cycles in the first case, and ~ 65 in the second case, which is inconsequential in most situations.

5 The AES-GCM-SIV Mode of Operation

We are now finally ready to formally present AES-GCM-SIV. As we have mentioned, AES-GCM-SIV differs from GCM-SIV in the use of POLYVAL, in the way the counter is defined in encryption, and due to the use of key derivation in every encryption. As such, AES-GCM-SIV can be viewed as a two-step procedure: a derivation of a per-nonce key from the master key, followed by the application of GCM-SIV$^+$using the derived keys. The algorithm is described in Fig. 3.

AES-GCM-SIV(keylength, K, N, AAD, MSG)

```
Key: K (Master Key)
Context: encryption-keylength (= 128 or 256)
         If keylength=128, AES is AES128, else AES is AES256
Input: AAD, MSG, N (96 bits)
Encrypt:
   (Record_Hash_key, Record_Enc_key) = DeriveKey(K, N)
   (C, TAG) = GCM-SIV+(Record_Hash_key, Record_Enc_key, N, AAD, MSG)
Output: (C, TAG)
```

Fig. 3. An outline of AES-GCM-SIV.

5.1 The Security of AES-GCM-SIV

In order to derive the exact security bounds for AES-GCM-SIV, we can simply combine the bound for GCM-SIV$^+$ in Theorem 5 together with the key derivation bound in Lemma 6. However, using a naive multi-key composition will yield bounds that are not very satisfactory in some settings. We therefore rely on the analysis carried out in [12] who prove very strong security bounds for AES-GCM-SIV. The following is proven in [12]:

Theorem 7 (AES-GCM-SIV bounds – Theorem 6.2 of [12]). *Assume the standard bounds regarding the indistinguishability of AES from a random permutation and a random function. Then, for every nonce adversary $\mathcal{A}$ attacking AES-GCM-SIV using $Q \leq 2^{64}$ different nonces, querying the ith nonce with N_E^i*

[3] If keys are derived by running $AES_K(i)$ for $i = 1, 2, \ldots$, then after 2^{64} key derivations, the derived keys cannot be argued to be indistinguishable from random. This is because a truly random key derivation mechanism would provide some colliding keys.

encryption queries and N_D^i decryption queries, the number of blocks processed with the ith nonce in both encryption and decryption queries is B_i, the longest AAD is less than 2^a blocks and the longest message is less than 2^m blocks, we have that $\mathcal{A}$'s advantage in the nonce-misuse resistance experiment is at most

$$\frac{3Q}{2^{96}} + \frac{Q^3}{3 \cdot 2^{2\kappa}} + \frac{Q \cdot B\mathsf{max}^2}{2^{129}} + \frac{5T_E}{2^{\kappa+1}} + \frac{(2^a + 2^m) \cdot \sum_{i=1}^{Q} N_D^i}{2^{128}} + \frac{\sum_{i=1}^{Q}(N_E^i)^2}{2^{126-m}}.$$

where $B\mathsf{max} = \max\{B_i\}_{i=1}^{Q}$, κ is the key length, and T_E is the offline preprocessing time that the adversary uses to attack multikey AES.

As shown in [12], for $Q \leq 2^{64}$ and AES with key-size 128 or above, it holds that the $\frac{3Q}{2^{96}}$ term dominates $\frac{Q^3}{3 \cdot 2^{2\kappa}}$ and so the latter can be removed. Likewise, in many (if not most) applications, one can assume that $(2^a + 2^m) \cdot \sum_{i=1}^{Q} N_D^i < Q \cdot B\mathsf{max}^2/2$, and thus $\frac{Q \cdot B\mathsf{max}^2}{2^{n+1}}$ dominates $\frac{(2^a+2^m) \cdot \sum_{i=1}^{Q} N_D^i}{2^n}$. Finally, for all "reasonable" adversaries, we have that $\frac{5T_E}{2^{\kappa+1}}$ is very small, and so can be ignored. This gives us the following very simple bound.

Theorem 8 (Corollary 6.3 of [12] – informal). *Assume the standard bounds regarding the indistinguishability of AES from a random permutation and a random function. Then, for every nonce adversary $\mathcal{A}$ attacking AES-GCM-SIV using $Q \leq 2^{64}$ different nonces in both encryption and decryption queries, querying the ith nonce with N_E^i different messages, and querying any single nonce with at most $B\mathsf{max}$ blocks overall and maximum message-length 2^m for any single message, we have that $\mathcal{A}$'s advantage in the nonce-misuse resistance experiment is at most*

$$\frac{3Q}{2^{96}} + \frac{Q \cdot B\mathsf{max}^2}{2^{129}} + \frac{\sum_{i=1}^{Q}(N_E^i)^2}{2^{126-m}}.$$

5.2 Using AES-GCM-SIV Randomly Selected Nonces

In this section, we consider the case that AES-GCM-SIV is used with a randomly generated IV, rather than with nonces guaranteed to be distinct. Recall that with AES-GCM, it is not possible to encrypt more than 2^{32} messages, since this will yield a collision in the IV with probability over 2^{-32}. This is therefore very limited. When using a nonce-misuse resistant scheme, the results are fundamentally different. This is due to the fact that a small number of collisions will not cause any harm (beyond knowing if the same or a different message was encrypted when the IV repeated). Thus, excellent bounds can be obtained.

This can be analyzed using the following theorem:

Theorem 9 (Theorem 2 of [19].). *Let $2 \leq r \leq q \leq A$. Let q balls be thrown, one by one (independently) at random, into A bins. Let $MultiColl(A, q, r)$ denote the event (called r multi-collision) that there exists at least one bin that contains at least r balls. Denote $\mu(A, q, r) = \frac{q^r}{r! \times A^{r-1}}$. Then,*

$$\Pr[MultiColl(A, q, r)] \leq \frac{\binom{q}{r}}{A^{r-1}} \leq \mu(A, q, r) \tag{10}$$

(under the simplifying assumption that 3 (or 2) multi-collision are rare, and therefore, the number of repeating nonces does not change the counting in a significant way. A bound on this number can be derived, to support this assumption). By Theorem 9, the probability that at least one IV repeats at least 4 times out of Q randomly selected 96-bit values is at most $\mu(96, Q, 4) = Q^4/\left(24 \cdot 2^{288}\right)$. This value is smaller than $\frac{3Q}{2^{96}}$ (in the bound given in Theorem 8) for any $Q \leq 2^{64}$, and so can be ignored. We thus have that the exact same bounds of Theorem 8 holds also for the case of random IVs. Using this fact, it was further shown in [12] that the term $\frac{\sum_{i=1}^{Q}(N_E^i)^2}{2^{126-m}}$ in Theorem 8 is insignificant, and thus the bound is reduced to just

$$\frac{3Q}{2^{96}} + \frac{Q \cdot B\mathsf{max}^2}{2^{129}}.$$

Thus, whereas AES-GCM with a random IV is limited to just 2^{32} encryptions, with AES-GCM-SIV it is even possible to encrypt $Q = 2^{64}$ messages of length 2^{12} each, or $Q = 2^{48}$ messages of length 2^{20} each; see Table 1. This is very significant for applications like QUIC where random IVs are needed, or in the general case where simply keeping state is undesirable.

5.3 Clarifying the Concrete Bounds

In Table 1, we provide concrete examples of the bounds that we obtain. In this table, Q is the number of unique nonces used for both encryption and decryption, and each nonce is assumed to be repeated N_E^i times. The maximum plaintext length for any message is $2^m - 1$ bytes.

Consider the case of a passive attacker who is looking to exploit a collision, or distinguish outputs from random, and observes encrypted traffic. Then, $Q \times N_E^i$ would be the maximum number of messages encrypted (assume no AAD for simplicity), and the bounds given cover both confidentiality and indistinguishability. NIST requires, for AES-GCM with random nonces, that the probability of a collision (and thus a failure of confidentiality) is at most 2^{-32}, limiting AES-GCM to 2^{32} messages. This table shows that AES-GCM-SIV can be used to encrypt many more messages while still meeting this requirement.

So, for example, the first row of the table captures the case where 2^{55} encryptions (or decryptions) of 64KiB messages are performed and, for arbitrary reasons, a nonce may end up being repeated 1,024 times. (This would take over a year at a rate of one billion encryptions per second.) Despite this, AES-GCM-SIV still meets the NIST requirements.

This example also shows that AES-GCM-SIV can encrypt a larger amount of data than AES-GCM. AES-GCM is limited to 2^{32} messages of 2^{32} blocks, giving 2^{68} bytes total. In this case, AES-GCM-SIV is encrypting $2^{45} \times 2^{10} \times 2^{16} = 2^{71}$ blocks (i.e., 2^{75} bytes).

An active attacker can submit attempted forgeries with a chosen nonce and length (of message and AAD). For example, he can clearly choose nonces in a way that (artificially) exceeds the limit on the number of repeats of a nonce (N_E^i). Of course, a collision may occur in this case. However, that collision is overwhelmingly likely to happen between two of the *attacker's* messages—gaining him nothing.

Scheme	Q	N_E^i	2^m	$\frac{Q \cdot B\text{max}^2}{2^{129}}$	$\sum_{i=1}^{Q} \frac{(N_E^i)^2}{2^{126-m}}$
AES-GCM-SIV (nonce)	2^{45}	2^{10}	2^{16}	2^{-32}	2^{-45}
	2^{32}	2^{15}	2^{16}	2^{-35}	2^{-48}
	2^{25}	2^{6}	2^{30}	2^{-32}	2^{-59}
	2^{32}	1	2^{30}	2^{-35}	2^{-62}
	1	2^{31}	2^{16}	2^{-34}	2^{-47}
	2^{42}	2^{8}	2^{16}	2^{-39}	2^{-52}
	2^{64}	2^{10}	2^{3}	2^{-39}	2^{-39}
	2^{64}	2^{15}	1	2^{-33}	2^{-31}
	2^{64}	2^{8}	2^{8}	2^{-33}	2^{-38}
	2^{48}	2^{10}	2^{14}	2^{-33}	2^{-44}
	2^{48}	2^{8}	2^{16}	2^{-33}	2^{-46}
	2^{64}	2^{10}	2^{10}	2^{-25}	2^{-32}
	2^{48}	2^{10}	2^{16}	2^{-29}	2^{-42}
	2^{32}	2^{10}	2^{24}	2^{-29}	2^{-50}
AES-GCM-SIV (random nonce)	2^{64}	-	2^{12}	2^{-35}	-
	2^{48}	-	2^{20}	2^{-35}	-

Table 1. Example parameters and security bounds for dominant terms (exponent rounded to nearest integer). Recall that Q is the number of different nonces in encryption and decryption queries, N_E^i is the number of messages encrypted per nonce (we assume all are equal), and $2^m - 1$ is the maximum message length. Observe that $B\text{max} = (N_E^i + N_D^i) \cdot 2^m$ when all messages are of maximum length. Bounds that are unacceptable are colored in red. See more explanations in the text.

6 Performance Results

This section provides some performance numbers. The measurements were taken on the microarchitecture codename "Skylake" (single core; Intel Turbo Boost Technology, Intel Hyper-Threading Technology, and Enhanced Intel Speedstep Technology disabled). On this processor, the latency and throughput of the AES and PCLMULQDQ instructions are, 4 and 1 cycles, respectively.

Fig. 4 shows the performance of POLYVAL and GHASH. As can be clearly seen, *POLYVAL* is always faster than *GHASH*, and tends to 20% faster as messages grow. This is therefore significant.

# of bytes	GHASH	POLYVAL	Speedup
	cycles	cycles	(POLYVAL over GHASH)
16	49	36	1.36
256	119	109	1.09
512	212	184	1.15
2,048	765	640	1.20
4,096	1,507	1,252	1.20

Fig. 4. The performance of POLYVAL and GHASH for different input lengths. Measurements were taken on the microarchitecture codename "Skylake". The GHASH numbers reflect the implementation of OpenSSL (1.0.2k).

Fig. 5 compares the performance of AES-GCM-SIV and GCM-SIV$^+$, for different message lengths, and for key sizes 128 and 256 bits. These measure our optimized code. In addition, we show the performance of AES-GCM for the same lengths. The AES-GCM code that we measured is the OpenSSL (1.0.2k) implementation, using the OpenSSL `speed` utility[4]; the results were converted to cycles and cycles per-byte (C/B). Note that this utility does not include the Init step. As a result (due to the structure of the OpenSSL code), this also means that the encryption of the mask first counter block (`100000000000000000000000000000000`) is not measured[5]. Therefore, to make a consistent comparison, the OpenSSL results needed to be adjusted by adding the cost of one encryption. We used a very generous estimation as follows: for the 128 bit case, we added 45 cycles, and for the 256 bit case, we added 60 cycles. These adjustments have negligible impact for long messages, but are noticeable for short ones. They are incorporated in the results shown in Fig. 5.

Message Length		AES-GCM-SIV	GCM-SIV$^+$	AES-GCM	AES-GCM-SIV	GCM-SIV$^+$	AES-GCM
(bytes)		128-bit key	128-bit key	128-bit key	256-bit key	256-bit key	256-bit key
		ENC / DEC	ENC / DEC	ENC / DEC	ENC / DEC	ENC / DEC	ENC / DEC
16	cycles	257 / 358	129 / 133	129 / 141	306 / 445	152 / 194	154 / 201
64	cycles	361 / 456	261 / 227	193 / 190	441 / 546	292 / 305	219 / 215
1,024	C/B	1.37 / 1.17	1.25 / 0.94	0.84 / 0.79	1.69 / 1.48	1.53 / 1.22	1.1 / 1.05
2,048	C/B	1.14 / 0.88	1.09 / 0.76	0.76 / 0.71	1.43 / 1.16	1.36 / 1.03	1.00 / 0.97
4,096	C/B	1.04 / 0.76	1.01 / 0.71	0.68 / 0.67	1.31 / 1.03	1.26 / 0.96	0.93 / 0.92
8,192	C/B	0.98 / 0.69	0.97 / 0.66	0.66 / 0.65	1.24 / 0.95	1.22 / 0.92	0.91 / 0.9
16,384	C/B	0.96 / 0.66	0.95 / 0.65	0.64 / 0.64	1.21 / 0.92	1.20 / 0.9	0.89 / 0.89

Fig. 5. The performance of AES-GCM-SIV, GCM-SIV$^+$, and AES-GCM (with a random 96-bit IV), with 128-bit and 256-bit keys. The measurements were taken on the micro-architecture codename "Skylake". The performance numbers are in processor cycles for short messages (16 and 64 bytes), and in cycles per byte (C/B) for long messages (1KB-16KB). See explanations in the text.

As can be seen in Fig. 5, for large messages, AES-GCM-SIV and GCM-SIV$^+$ decryption is have comparable decryption times to AES-GCM. Furthermore, encryption is approximately 50% more expensive that AES-GCM. This is due to the fact that full nonce-misuse resistance requires two passes. Nevertheless, for messages of 8KB and over with AES-128, the time is still under 1 C/B. It is also clear that the addition of the key derivation in AES-GCM-SIV (over GCM-SIV$^+$) is inconsequential for large messages. For relatively short messages of length 1KB, the key derivations adds only about 10%. Given the improved bounds achieved by this derivation, we find the tradeoff very favorable.

In contrast, we do note that for very short messages (e.g., 16 bytes), the additional cost in percentage is very considerable. In typical applications, for such short messages, the difference in actual time is hardly measurable. However, when this is not the case, and the number of overall encryptions stays within acceptable bounds, then GCM-SIV$^+$ can be used.

7 Remarks and Conclusions

The goal of achieving fast nonce misuse resistant authenticated encryption is an important one. The GCM-SIV mode of operation [11] achieves very high performance. However, its security bounds are not optimal. In particular, when encrypting many short messages, or when a nonce repeats very frequently, the concrete security margins degrade. The SIV [15] and GCM-SIV [11] schemes both have a security bound that is typically dominated by $q^2/2^{n-k}$, where q is the overall number of encryptions.[6] Our primary aim in designing

[4] For example, `openssl speed -evp aes-128-gcm`, and `openssl speed -decrypt -evp aes-256-gcm`.

[5] Technically, the speed utility measures AES-GCM with a fixed key and repeating nonces, which does not really represent a legitimate usage of the cipher, rather a performance characteristic.

[6] We remark that in the SIV scheme, 2^k is the longest message encrypted in actuality, whereas in GCM-SIV 2^k is an a priori bound on the maximum size of the message. In this sense, SIV and GCM-SIV$^+$ have essentially the same bound of approximately $q^2/2^{n-k}$.

AES-GCM-SIV (over the original GCM-SIV) was to provide better bounds, as required in some applications where encryptions are issued at a very high rate and nonces may repeat. AES-GCM-SIV achieves a significantly better bound than SIV and GCM-SIV, reducing the quadratic dependency on the number of queries to an almost linear dependency. As can be seen in Theorem 8, the dominating term for most parameters is $\frac{Q \cdot B\mathsf{max}^2}{2^{129}}$ where Q equals the number of different nonces used in encryption and decryption queries, and $B\mathsf{max}$ is the maximum number of blocks processed with any given nonce. (Of course, there are other terms in the bound, but this dominates in many cases.)

An interesting artifact of the security bound that we derive is that the security margin obtained depends on how many times the nonce repeats. To go to the extreme, if the same nonce is used always then we have that $Q = 1$ and N_E^1 is the total number of encrypted message. In this case, $B\mathsf{max} = N_E^1 \cdot 2^m$. In this case, if $m = 2^{20}$ and $N_E^1 = 2^{44}$, then $\frac{Q \cdot B\mathsf{max}^2}{2^{129}} = \frac{2^{128}}{2^{129}} = \frac{1}{2}$ which is not secure. This should be contrasted with the case that a different nonce is used always. In such a case, when encrypting 2^{44} messages of length 2^{20} each, we have that $Q = 2^{44}$ and $B\mathsf{max} = 2^{20}$ and so $\frac{Q \cdot B\mathsf{max}^2}{2^{129}} = \frac{2^{84}}{2^{129}} = 2^{-45}$, which is not even close to the limit.

When discussing this work, we found a widespread misunderstanding of the term "nonce-misuse resistant". Many people appear to expect the security of a nonce-misuse resistant scheme to be completely unaffected by the number of times that a nonce is reused. Thus, while it is a convenient shorthand to distinguish schemes that tolerate repeated nonces (e.g., AES-GCM-SIV) from those that do not (e.g., AES-GCM and ChaCha20-Poly1305), nonce-misuse resistance is not necessarily a "binary" property. In particular, it is not binary for the case of AES-GCM-SIV, and as we have shown the security bounds of AES-GCM-SIV change as the number of repeated nonces varies.

As such, it is important to understand what security is actually guaranteed by nonce-misuse resistance. First and foremost, nonce-misuse resistant schemes reveal when the same plaintext is encrypted using the same nonce, and this is well understood. Due to this, some have concluded that if an application guarantees unique plaintexts, then the same nonce can be safely reused in every encryption. Although this is true in some sense, it is also true that the security bounds can be degraded, as we have shown in this paper. Thus, it is not recommended to purposefully use AES-GCM-SIV with the same nonce (unless the number of encryptions is small enough so that the quadratic bound is small). We stress that if the same nonce is used always, then AES-GCM-SIV is no worse than previous schemes; however, AES-GCM-SIV can achieve far better bounds and it is worth taking advantage of this.

We believe that AES-GCM-SIV is well suited to applications where independent servers need to work with the same key, and where nonce repetition is a real threat. In such cases, it is not possible to enjoy the better bounds available for encryption schemes that utilize state to ensure unique nonces in every encryption. Encryption of TLS session tickets and QUIC's source-address tokens are good examples of use cases that are particularly well suited, because all large deployments of TLS and QUIC will involve multiple, geographically-separated servers encrypting with a common key. However, we remark that AES-GCM-SIV provides excellent bounds for all sets of parameters and so is a good choice in general. Also, as we have shown, its performance is extremely good on processors with AES-NI instructions.

We conclude by remarking that applications that encrypt files with random keys, and wrap those keys with a master key, are another example where AES-GCM-SIV is well suited. The key-wrapping step involves encrypting very short messages (just one or two blocks for a symmetric key, possibly with one additional AAD block) with a common key. In this situation $m < 2$ and thus Theorem 8 suggests that using a *fixed nonce* with AES-GCM-SIV provides comfortable security bounds, even for very large numbers of wrapped keys. Specifically, since $B\mathsf{max} < N_E \cdot 4$ where N_E denotes the total number of encryptions, the security bound is dominated by $\frac{16 {N_E}^2}{2^{126-m}} = \frac{{N_E}^2}{2^{120}}$, which is acceptable even up to $N_E = 2^{44}$. In this case, one might wish to cache the result of the KDF. We remark that this is the same as using GCM-SIV$^+$ directly, with a fixed nonce.

Technical comments. The code used for the performance measurements in this paper can be found at https://github.com/Shay-Gueron/AES-GCM-SIV, and is available for general use. The current version of BoringSSL also supports AES-GCM-SIV. The latest version of the CFRG specification can always be found at https://tools.ietf.org/html/draft-irtf-cfrg-gcmsiv. At the time of writing, the most current version is version five.

Acknowledgments

This research was supported by the PQCRYPTO project, which was partially funded by the European Commission Horizon 2020 research Programme, grant #645622, by the ISRAEL SCIENCE FOUNDATION (grant No. 1018/16), and by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office.

References

1. BoringSSL, https://boringssl.googlesource.com/boringssl/
2. RFC5077: Transport Layer Security (TLS) Session Resumption without Server-Side State, https://tools.ietf.org/html/rfc5077#section-4
3. A. Abdalla and M. Bellare. Increasing the Lifetime of a Key: A Comparative Analysis of the Security of Re-keying Techniques. In *ASIACRYPT 2000*, Springer (LNCS 1976), pages 546–559, 2000.
4. M. Bellare and B. Tackmann. The Multi-User Security of Authenticated Encryption: AES-GCM in TLS 1.3. In *CRYPTO 2016*, Springer (LNCS 9814), pages 2470276, 2016.
5. M. Dworkin. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) for Confidentiality and Authentication. *Federal Information Processing Standard Publication* FIPS 800-38D, 2006. http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf
6. S. Gilboa and S. Gueron. How many queries are needed to distinguish a truncated random permutation from a random function?. To appear in the *Journal of Cryptology*.
7. S. Gilboa and S. Gueron. The Advantage of Truncated Permutations. Manuscript, 2016. https://arxiv.org/abs/1610.02518.
8. S. Gueron, AES-GCM-SIV, https://github.com/Shay-Gueron/AES-GCM-SIV
9. S. Gueron, A new interpretation for the GHASH authenticator of AES-GCM. *manuscript*, 2017.
10. S. Gueron, A. Langley and Y. Lindell. AES-GCM-SIV: Nonce Misuse-Resistant Authenticated Encryption. *CFRG Draft*, 2016. https://tools.ietf.org/html/draft-irtf-cfrg-gcmsiv
11. S. Gueron and Y. Lindell. GCM-SIV: Full Nonce Misuse-Resistant Authenticated Encryption at Under One Cycle per Byte. In the *22nd ACM Conference on Computer and Communications Security* (CCS), page 109–119, 2015.
12. S. Gueron and Y. Lindell. Better Bounds for Block Cipher Modes of Operation via Nonce-Based Key Derivation. *Manuscript*, 2017.
13. D.A. McGrew and J. Viega The Galois/Counter Mode of Operation (GCM). http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf
14. D.A. McGrew and J. Viega The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In *INDOCRYPT 2004*, Springer (LNCS 3348), pages 343–355, 2004.
15. P. Rogaway and T. Shrimpton. Deterministic Authenticated Encryption: A Provable-Security Treatment of the Key-Wrap Problem. In *EUROCRYPT 2006*, Springer (LNCS 4004), pages 373–390, 2006.
16. P. Rogaway and T. Shrimpton. The SIV Mode of Operation for Deterministic Authenticated-Encryption (Key Wrap) and MisuseResistant Nonce-Based Authenticated-Encryption. Available from http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/siv/siv.pdf (2007)
17. QUIC, a multiplexed stream transport over UDP. https://www.chromium.org/quic
18. A. J. Stam, Distance between sampling with and without replacement, Statist. Neerlandica **32** (1978), no. 2, 81–91.
19. K. Suzuki, D. Tonien, K. Kurosawa and K. Toyota. Birthday Paradox for Multi-collisions. Proceedings of the *9th International Conference on Information Security and Cryptology*, Springer (LNCS 4296), pages 29–40, 2006.