GCM-SIV: Full Nonce Misuse-Resistant Authenticated Encryption at Under One Cycle per Byte*

Shay Gueron† Yehuda Lindell‡

July 12, 2017

Abstract

Authenticated encryption schemes guarantee both privacy and integrity, and have become the default level of encryption in modern protocols. One of the most popular authenticated encryption schemes today is AES-GCM due to its impressive speed. The current CAESAR competition is considering new modes for authenticated encryption that will improve on existing methods. One property of importance that is being considered more today – due to multiple real-life cases of faulty sources of randomness – is that repeating nonces and IVs can have disastrous effects on security. A (full) nonce misuse-resistant authenticated encryption scheme has the property that if the *same* nonce is used to encrypt the *same* message twice, then the same ciphertext is obtained and so the fact that the same message was encrypted is detected. Otherwise, *full security* is obtained – even if the same nonce is used for different messages.

In this paper, we present a new fully nonce misuse-resistant authenticated encryption scheme that is based on carefully combining the GCM building blocks into the SIV paradigm of Rogaway and Shrimpton. We provide a full proof of security of our scheme, and an optimized implementation using the AES-NI and PCLMULQDQ instruction sets. We compare our performance to the highly optimized OpenSSL 1.0.2 implementation of GCM and show that our *nonce misuse-resistant* scheme is only 14% slower on Haswell architecture and 19% slower on Broadwell architecture. On Broadwell, GCM-SIV encryption takes only *0.92 cycles per byte*, and GCM-SIV decryption is exactly the same as GCM decryption taking only 0.77 cycles per byte. In addition, we compare to other optimized authenticated-encryption implementations carried out by Bogdanov et al., and conclude that our mode is very competitive. Beyond being very fast, our new mode of operation uses the same building blocks as GCM and so existing hardware and software can be utilized to easily deploy GCM-SIV. We conclude that GCM-SIV is a viable alternative to GCM, providing full nonce misuse-resistance at little cost.

1 Introduction

Authenticated encryption. A symmetric encryption scheme achieves authenticated encryption

*An extended abstract of this work appeared at *ACM CCS 2015*. In previous versions of this paper, we forgot to add the factor of $\left(\lceil \frac{L}{n} \rceil + 1\right)$ that is multiplied by $q_E(\mathcal{A})^2$ in Eq. (4), stating the bound for the 2-key GCM-SIV version. We thank Tetsu Iwata and Yannick Seurin for pointing this out.

†Department of Mathematics, University of Haifa and Intel Corporation, Israel. shay@math.haifa.ac.il. Supported by the PQCRYPTO project, which was partially funded by the European Commission Horizon 2020 Research Programme, grant #645622.

‡Department of Computer Science, Bar-Ilan University, Israel. lindell@biu.ac.il. Supported by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minster's Office.

if it provides both privacy and integrity. Informally, such a scheme provides the guarantee that no adversary can generate a ciphertext that decrypts to a valid value, and that encryptions of plaintexts are indistinguishable from each other. Classically, authenticated encryption was achieved via the "encrypt-then-authenticate" paradigm that states that one should first encrypt and then apply a message authentication code to the obtained ciphertext [5, 13]. This methodology is sound, but is often inefficient. A more general study of composition methods, considering multiple different options and security goals, was carried out in [18]. However, in many cases, dedicated modes of encryption have been proposed that are optimized for high performance. One of the most popular such modes used today is GCM, and it has very fast performance on the latest Intel architectures. For example, it achieves performance of 0.77 cycles per byte (C/B hereafter) on the architecture codename Broadwell.[1] Many authenticated encryption modes have been proposed, and the CAESAR competition currently being run is considering some of them.

IV and nonce misuse resistance. For a long time, authenticated encryption was considered the highest level of security for symmetric encryption schemes. Indeed, with respect to adversarial capabilities, this is the case. However, it has been observed that in many cases, something can go wrong in the encryption procedure. For example, when random IVs are needed and encryption is carried out on devices with weak randomness (like mobile phones), the IV may repeat with high probability. This same problem arises on strong devices, where the random source is buggy. Likewise, in nonce-based encryption schemes (where the requirement is just that a unique nonce is used every time), the repetition of a nonce happens in practice and compromises security.[2] In order to see why repeating IVs or nonces can have disastrous results, consider the case that the counter is repeated in counter-mode encryption. In this case, all security is lost (an attacker can easily detect that this is the case, and can just XOR the ciphertexts in order to obtain the XOR of the plaintexts). In another example, in GCM encryption, if the initial counter is repeated, then this also completely compromises the integrity property and an attacker who views the two encryptions can generate as many forgeries as it wishes in the future.

These observations prompted Rogaway and Shrimpton in a breakthrough work to define the notion of nonce misuse-resistant authenticated encryption [20]. An encryption scheme with this property provides an extraordinarily high level of security. If the same nonce is used to encrypt the same message, then this will be detected by the adversary (since the same output will be obtained both times). Otherwise, full security will be obtained. This means that encrypting different messages with the same nonce will reveal nothing whatsoever (except for the fact that the messages were different). Such an encryption scheme can be used comfortably in scenarios where devices cannot be trusted to generate good quality randomness. Indeed, it is advisable to always use misuse resistant encryption, since low quality randomness has shown up multiple times due to software bugs. Unfortunately, nonce misuse-resistant authenticated encryption is inherently less efficient since it requires two passes over the data.

We remark that a weaker notion of nonce misuse-resistance has been proposed, where some more information is allowed to be revealed in the case of a nonce repeating. Specifically, if a nonce repeats between two messages that have a common prefix, then this fact is revealed, along with the length of the common prefix [6]. This definition allows for achieving support for *online encryption*,

[1] Broadwell (and similarly Haswell) is an Intel Architecture Codename of a very recently announced micro-architecture. Broadwell is the 5th Generation Intel® Core Processor, and Haswell is the 4th Generation Intel® Core™ Processor. They can have different configurations in different products. Hereafter, for short, we refer to them simply as Broadwell and Haswell (or BDW/HSW).

[2] In this paper, we refer to IVs as values that must be *randomly* chosen, whereas nonces must simply be non repeating.

where the encryptor does not need to hold the entire plaintext at any time (this is impossible to achieve under the more stringent notion of [20]). In this paper, we adopt the stronger notion.

Our contributions. In this paper, we present a new fast mode of encryption that achieves *nonce misuse-resistant authenticated encryption*. Our mode of encryption is based on the GCM mode of encryption, and a careful combination of the building blocks used in GCM together with the SIV construction paradigm of [20]; we call this mode GCM-SIV. Our mode of operation works by first applying GHASH (the GCM authenticator) to the plaintext and any associated authenticated data, and then applying a pseudorandom function to the result XORed with the IV. We show that this is a pseudorandom function over the nonce, plaintext and associated data, and thus the result can be used as the IV in CTR mode. Therefore, this is a highly efficient instantiation of the IV misuse resistance mode proven in [20] (and further abstracted in [18]).

Beyond a full specification of the scheme and proof of security, we provide optimized implementations of GCM-SIV and compare their performance to GCM on the Haswell and Broadwell Intel architectures. We compare our performance to the *highly optimized OpenSSL (v.1.0.2) implementation of GCM* on the same architectures, and provide exact cycle counts. (We argue that such a detailed study is essential to truly understand the efficiency of new modes of encryption.)

The advantages of our construction are:

1. **Ease of deployment:** Our construction uses the same building blocks as GCM. Therefore, existing code bases (and hardware implementations) can be used to implement GCM-SIV. This is an important consideration when adopting new modes of operation. In addition, our construction only uses AES encryption (and not its inverse).

2. **Encryption performance on Intel architectures:** Encryption under GCM-SIV is not far from the performance of GCM. For encryption, GCM-SIV is only 14% (resp., 19%) slower than GCM on Haswell (resp., Broadwell). This difference is due to the fact that using the Intel AES-NI and PCLMULQDQ instructions, the GHASH and AES operations can be run in parallel in GCM, but must be computed one after the other in GCM-SIV.

   Concretely, our implementation runs at 1.17 cycles per byte on the Haswell architecture, and 0.92 cycles per byte on the recent Broadwell architecture. Thus, we obtain full nonce misuse-resistance on the latest Intel architecture *with encryption costing less than one cycle per byte*!

3. **Encryption performance on other architectures:** GCM-SIV is fast on *every* architecture that has support for AES-NI and carry-less multiplication, in some form. The current trend is that such support is offered on most (if not all) the new 64-bit architectures. Some examples are AMD (Bulldozer) and NVidia (Denver), ARM little core (A53) and big core (A57), and Apple (A7/A8). With this ubiquitous support, AES-GCM (and hence our GCM-SIV) would enjoy the best performance on most modern platforms.

   We remark that on architectures where AES-NI is not available (and a single thread is used), the cost of GCM-SIV is the same as GCM alone. This is due to the fact that the operations carried out in GCM-SIV are almost identical to that of GCM, and on such architectures the computation of GCM cannot be parallelized with a single thread.

4. **Decryption performance:** Decryption of GCM-SIV has the *exact* same cost as decryption of GCM. This is due to the fact that they have the same operations and in decryption they can both be parallelized. Both achieve rates of 0.77 cycles per byte on Broadwell.

5. **Encryption of short messages:** For short messages (up to 32 bytes), GCM-SIV is actually *more efficient* than GCM. Thus, it is preferable for key wrapping and in settings were many short messages are sent.

In summary, GCM-SIV achieves *full* nonce misuse-resistant authenticated encryption at an extremely low cost. It has a proof of security, and a full implementation to validate its true cost. Finally, it is easily deployable due to existing hardware support on a wide range of processors.

Comparison to other full nonce-misuse resistant schemes. We compare GCM-SIV to other schemes that provide the same level of nonce-misuse resistance. Clearly, GCM-SIV is much faster than the original SIV of [20] since it uses CMAC which is not parallelizable. Therefore, on a platform using AES-NI, the performance is dominated by the latency of the AES-NI (7 cycles on Broadwell/Haswell), and is 4.44 cycles per byte. Therefore, the performance of the original SIV scheme would be at least 5 cycles per byte on Broadwell/Haswell.

The HBS and BTM modes of operation [14, 15] have a similar *theoretical complexity* to our implementation. However, our scheme is far simpler, on small messages our mode is faster, and importantly we can reuse GCM software and hardware which are already widely deployed. Finally, we have a fully optimized implementation to validate our claims of efficiency, whereas we are not aware of such results for HBS and BTM.

Of the CAESAR competition candidates, the only three that achieve full nonce misuse-resistance are AEZ [12], Julius [2] and HS1-SIV [16]; see also [1]. On Broadwell, AEZ can achieve (for long messages) a throughput of 0.7 C/B [12]. This relative performance advantage of AEZ over GCM-SIV exists for encryption only, and not decryption. In addition, AEZ meets a slightly stronger form of security, called *robust* authenticated encryption. However, this comes at the cost of relying on a nonstandard security assumption; specifically, AEZ internally uses AES with just 4 rounds and so is not a full block cipher operation. Julius (ECB/CTR) requires 1 AES computation plus 2 field multiplications per block. An optimized software implementation of Julius using the AES-NI instruction set was carried out by [3] as discussed below. As can be seen in Table 1, it is significantly slower than GCM-SIV.

HS1-SIV takes a different approach, and is targeted at achieving good performance on platforms that do not have the AES-NI/PCLMULQDQ instructions (e.g., embedded systems). ChaCha20/Poly1305 is another authenticated encryption scheme that targets performance on CPUs that have no AES-NI/PCLMULQDQ. It is gaining popularity on small-device client platforms. However, on platforms with AES-NI/PCLMULQDQ (e.g., servers that service such connections and the latest mobile devices that do have strong CPUs with these instructions), ChaCha20/Poly1305 (stream cipher and authenticator) is outperformed by GCM-SIV (and AES-GCM) due to the dedicated hardware support. Optimized ChaCha20 (i.e., encryption alone) consumes approximately 1.04 cycles per byte on Broadwell, and the fastest Poly1305 implementations we are aware of performs at 0.67/ 0.66 C/B on HSW/BDW.This is significantly slower than GCM-SIV on these platforms.

In addition to the above, [19] recently presented a nonce misuse-resistant version of OMD. Their scheme requires 2 AES operations per block, and so would cost at least 1.3 cycles per byte using optimized code and AES-NI.

Comparison to nonce-misuse AE modes on HSW. In a recent publication [3], Bogdanov et al. report on optimized AES-NI implementations of authenticated-encryption modes of operation on Intel Haswell architecture. In Table 1, we show the results for the nonce misuse-resistant schemes and compare them to ours. We note that amongst these modes, only Julius receives full

Mode	# msgs.	Message length (bytes) 128	256	512	1024	2048
		single message				
McOE-G		7.77	7.36	7.17	7.07	7.02
COPA		**3.37**	**2.64**	**2.27**	**2.08**	**1.88**
POET		5.30	4.93	4.75	4.68	4.62
Julius		4.18	4.69	3.24	3.08	3.03
	# msgs.	multiple messages				
McOE-G	7	1.91	1.76	1.68	1.64	1.62
COPA	15	**1.62**	**1.53**	**1.48**	**1.46**	**1.45**
POET	8	3.24	3.24	2.98	2.79	2.75
Julius	7	2.53	2.27	2.16	2.09	2.06
		single message				
GCM-SIV		**2.20**	**1.66**	**1.41**	**1.28**	**1.22**

Table 1: Performance comparison of GCM-SIV (bottom row) to the AES-NI optimized implementations of nonce misuse-resistant schemes, reported in [3] (top rows). The measurements are on a Haswell processor and are in cycles per byte.

nonce misuse-resistance; the others achieve online encryption and thus the weaker notion of misuse resistance. The optimized implementations by [3] are for a single message and for multiple messages processed in parallel; see Table 1.

Observe that GCM-SIV is faster than Julius for all message sizes, and even when considering the speed for parallel multiple messages for Julius versus a single message for GCM-SIV. The fastest mode shown by [3] is COPA [4]. Observe that GCM-SIV is significantly faster than COPA for all message sizes when processing a single message. In addition, for *large messages* (of size greater than 512 bytes), GCM-SIV outperforms COPA, even when comparing parallel multiple messages for COPA to a single message for GCM-SIV. (In contrast, for short messages COPA is up to 25% faster for multiple messages. However, recall that GCM-SIV achieves a higher level of security than COPA, and COPA is only faster when processing multiple messages in parallel.)

This comparison sheds significant light on the efficiency of GCM-SIV since it compares it to highly optimized implementations of analogous modes on exactly the same architecture.[3]

Organization. We use the notions of CPA-secure IV-based encryption (ivE), nonce-based authenticated encryption (nAE), and nonce misuse-resistant authenticated encryption (mrAE), as defined in [20, 18]. These definitions are repeated in Appendix A for the sake of completeness. In Section 2 we describe the abstract SIV scheme of Rogaway and Shrimpton [20], and in Section 3 we present and prove the security of our specific instantiation based on any XOR universal hash function and any pseudorandom function. Our proof includes a concrete analysis and bounds. In Section 4, we describe the final concrete scheme that uses the GHASH universal hash function (from GCM) and AES. Finally, in Section 5 we provide an in-depth analysis of the performance of our scheme.

[3] We stress that McOE-G, COPA and POET do not achieve full misuse resistance, and only achieve a weaker notion called "online authenticated encryption". However, they do enable online encryption with a single pass and constant memory, unlike any full misuse-resistant scheme. Nevertheless, the comparison is helpful to understanding the performance of GCM-SIV.

Our analysis includes an exact operation count, along with a description of our empirical results. We provide actual cycle counts for different size messages on Haswell and Broadwell, and compare them to the actual cycle counts of the optimized OpenSSL (v.1.0.2) implementation of GCM.

2 The Abstract SIV Encryption Scheme

In [18], a number of constructions for authenticated encryption were considered. The construction called A4 is a generalisation of the SIV mode of operation [20] that has been proven to be nonce-misuse resistant. In this section, we describe this abstraction.

Let $F_{K_1} : \{0,1\}^* \to \{0,1\}^n$ be a pseudorandom function and let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a CPA-secure IV-based encryption scheme. For simplicity, we assume that all keys are of length n. Let k be a parameter such that the maximum message length is $2^k \cdot n$ bits.

Let Encode be a function that outputs a unique encoding of its (multiple) inputs as a string. Thus $\mathsf{Encode}(x, y)$ fully determines the pair of inputs x and y. We remark that simply concatenating x with y is not a unique encoding, since this would imply that $\mathsf{Encode}(x, y) = \mathsf{Encode}(x \| y_1, y_2)$ where $y = y_1 \| y_2$ and $\|$ denotes concatenation.

Construction 2.1

- **Key generation:** $K_1, K_2 \in \{0,1\}^n$ *chosen uniformly and independently at random*
- **Encryption of M with associated data A and nonce N:**
  1. *Compute* $T = F_{K_1}(\mathsf{Encode}(N, A, M))$
  2. *Let* IV *be the* $n - k$ *most significant bits of* T,[4] *and compute* $C = \mathsf{Enc}_{K_2}(IV, M)$
  3. **Output:** (N, A, C, T)
- **Decryption of (N, A, C, T):**
  1. *Let* IV *be the* $n - k$ *most significant bits of* T.
  2. *Compute* $M = \mathsf{Dec}_{K_2}(IV, C)$
  3. *Compute* $T' = F_{K_1}(N, A, M)$
  4. **Output:** *If* $T' = T$ *then output* (A, M)*; else output* $\perp$

Security. In [18], the following is proven (this construction is called A4 in [18]).

Theorem 2.2 (Proven in Section A.3 of [18]) *If* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is a secure IV-based encryption scheme and* F *is a pseudorandom function, then Construction 2.1, denoted* Π*, is a secure nonce-based authenticated encryption scheme.*

Concretely, [18, Section A.3] proves that the nAE advantage of any adversary $\mathcal{A}$ for this construction is:

$$\mathbf{Adv}^{\mathrm{nAE}}_{\Pi}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{B}(\mathcal{A})) + \mathbf{Adv}^{\mathrm{ivE}}_{\mathsf{Enc}}(\mathcal{D}(\mathcal{A})) + \frac{q_d}{2^n}$$

[4] This is not necessary in the abstract SIV scheme, but we will use this later on.

with the following parameters. Let n be the output length of the pseudorandom function. Let $\mathcal{A}$ ask q_E queries to its encryption oracle, and q_d queries to its decryption oracle, with $q = q_E + q_d$. The encryption queries have total length L_e, and the decryption queries have total length L_d, with $L = L_e + L_d$. Let $t_{\mathcal{A}}$ be the running time of $\mathcal{A}$, given this total number and length of queries, let $t_F(L)$ be the running-time to compute the pseudorandom function F on inputs of length L, and likewise $t_{\mathsf{Enc}}(L)$ for the underlying ivE encryption. Then reduction $\mathcal{B}$ has running time at most $t_{\mathcal{A}} + 2t_F(L) + t_{\mathsf{Enc}}(L)$, asks at most $2(q_E + q_d)$ queries to its oracle, with total length at most L. Reduction $\mathcal{D}$ has running time $t_A + t_F(L) + t_{\mathsf{Enc}}(L_e)$, asks at most q_E queries to its oracle, with total length at most L_e.

Theorem 2.2 considers nonce-based authenticated encryption. However, we have to prove nonce *misuse-resistant* authenticated encryption. In order to see that this holds, note that the security of the nonce-based authenticated encryption holds as long as the input to the pseudorandom function is different each time, since this guarantees pseudorandom output each time (up to the probability of a collision). In the nonce-based encryption setting this is guaranteed by always using a different nonce. However, since the pseudorandom function is applied to entire triple (N, A, M) in this construction, it receives a different input each time as long as the same (N, A, M) is not used twice. However, this is exactly what happens in the nonce misuse-resistant setting. Thus, we conclude that the exact same security and bounds are achieved in this setting. We conclude:

Corollary 2.3 *If* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is a secure IV-based encryption scheme and* F *is a pseudorandom function, then Construction 2.1, denoted* Π*, is a secure nonce misuse-resistant authenticated encryption scheme. In addition,*

$$\mathbf{Adv}^{\mathrm{mrAE}}_{\Pi}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{B}(\mathcal{A})) + \mathbf{Adv}^{\mathrm{ivE}}_{\mathsf{Enc}}(\mathcal{D}(\mathcal{A})) + \frac{q_d}{2^n}$$

where $\mathcal{B}$*,* $\mathcal{D}$*,* q_d *and* n *are as above.*

3 The Universal-SIV Instantiation

In this section, we describe our Universal-SIV construction, which is an instantiation of the abstract construction provided in Section 2 as follows:

- The pseudorandom function applied to the data (associated authentication data as well as plaintext) is constructed by computing a universal hash function on the data, XORing in the nonce, and then applying a pseudorandom function to the result.
- The encryption scheme used is CTR mode, where the initial counter is $n - k$ bits long and the remaining k bits in the block are used for counters for a message with at most 2^k blocks of length n. Note that this method ensures that as long as the same initial counter is not used twice, no counter in any block is reused.

3.1 The Universal-SIV Specification

The scheme uses the following primitives:

- An ϵ-XOR universal hash function $H_{K_1} : \{0,1\}^* \to \{0,1\}^n$. Formally, a hash function is ϵ-XOR universal if for every $x, y, z \in \{0,1\}^*$ it holds that $\Pr_{K_1}[H_{K_1}(x) \oplus H_{K_1}(y) = z] \leq \epsilon(n)$,

where the probability is over the choice of K_1. For simplicity, we assume that the key length is n.

- A pseudorandom function $F_{K_2} : \{0,1\}^n \to \{0,1\}^n$; for simplicity, we assume that the key length is n.
- A parameter $k < n$, where $2^k \cdot n$ is the maximum message length.
- The GCM encoding function $\mathsf{Encode}(x, y)$. This encoding works by adding an additional block of 128 bits, where the 64-bits contains the length of x and the second 64-bits contains the length of y.

Construction 3.1 (The Universal-SIV scheme for block length n):

- **Key generation:** *$K_1, K_2, K_3 \in \{0,1\}^n$ chosen uniformly and independently at random*
- **Encryption of M with associated data A and n-bit nonce N:**
  1. **Step 1:** *Compute $h = H_{K_1}(\mathsf{Encode}(A, M))$*
  2. **Step 2:** *Compute $T = F_{K_2}(h \oplus N)$*
  3. **Step 3:** *Encrypt M with CTR mode using the pseudorandom function F with key K_3. The initial counter is taken to be the $n-k$ most significant bits of T followed by k zeroes. Denote the initial counter by $I_1 \| I_2$ where $I_1 \in \{0,1\}^{n-k}$ and $I_2 = 0^k$; stated otherwise, the initial counter equals $I_1 \cdot 2^{n-k} + I_2$. The jth counter is defined to be $I \cdot 2^{n-k} + [(I_2 + j) \bmod 2^k]$ for $j = 0, \ldots, 2^k - 1$.*
     Denote the resulting ciphertext by C
  4. **Output:** (N, A, C, T)
- **Decryption of (N, A, C, T):**
  1. **Step 1:** *Decrypt C with CTR mode using F with key K_3, and using the $n-k$ most significant bits of T as the initial counter; denote the resulting plaintext by M.*
  2. **Step 2:** *Compute $h = H_{K_1}(\mathsf{Encode}(A, M))$*
  3. **Step 3:** *Compute $T = F_{K_2}(h \oplus N)$*
  4. **Output:** *If $T' = T$ then output (A, M); else output $\perp$.*[5]

We remark that for standard nonce-based authenticated encryption, it would suffice to take $T = h \oplus F_{K_2}(N)$. However, if T is computed in this way and a nonce N is repeated for two different messages with hash results h, h' then it is possible to XOR the two tags together and obtain $h \oplus h'$ (since the mask F_{K_2} disappears). In this case, the adversary obtains two messages and their hash, and can forge messages (since this suffices to learn the key K_1 for H). For this reason, we compute the tag as $T = F_{K_2}(h \oplus N)$. Formally, it is required that T be computed by applying a pseudorandom function to (N, A, M), as described in Construction 2.1.

[5] A constant-time comparison function must be used here.

3.2 Proof of Security of Universal-SIV

Notation. We provide a concrete analysis of security, counting the running time of the adversaries, the number of oracle queries that they make, and their advantage. For an adversary $\mathcal{A}$ we denote by $t(\mathcal{A})$ its running time, and by $q(\mathcal{A})$ the number of oracle queries it makes. For the sake of clarity, we differentiate between different types of oracle queries and denote by $q_E(\mathcal{A})$ the number of oracle queries to the encryption oracle (where such an oracle is given), by $q_D(\mathcal{A})$ the number of oracle queries to the decryption oracle (where such an oracle is given), and by $q_f(\mathcal{A})$ the number of oracle queries to the function oracle (for adversaries distinguishing a pseudorandom function from a random one). Finally, for a function F, we denote by $t_F(L)$ the time taken to compute F on overall inputs of length L.

Proof of security. By Corollary 2.3, in order to prove security we need to show that

$$\mathcal{F}_{K_1,K_2}(N\|M) \stackrel{\text{def}}{=} F_{K_2}(H_{K_1}(M) \oplus N)$$

is a pseudorandom function from $\{0,1\}^n \to \{0,1\}^n$, when H is an ϵ-XOR universal hash function from $\{0,1\}^* \to \{0,1\}^n$. (Note that M here includes an encoding of both the associated data and plaintext message used in the encryption process. We removed the explicit reference to A for clarity.)

Before proving that $\mathcal{F}$ is indeed a pseudorandom function, we define security for pseudorandom functions via the following experiment:

Experiment $\mathsf{Expt}^b_{\mathcal{A},\mathcal{F}}$

1. If $b = 0$ then choose K at random and set $\mathcal{O} = \mathcal{F}_K$.

   Else, if $b = 1$, set $\mathcal{O}$ to be a truly random function $f : \{0,1\}^* \to \{0,1\}^n$.
2. $b' \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}$
3. Output 1 if and only if $b' = b$

Definition 3.2 *A family of functions $\mathcal{F}$ is a (t, q_f, δ)-*pseudorandom function *if for every adversary $\mathcal{A}$ running in time at most t and asking at most q_f queries to its oracle it holds that,*

$$\mathbf{Adv}^{\text{prf}}_{\mathcal{F}}(\mathcal{A}) \stackrel{\text{def}}{=} \left|\Pr[\mathsf{Expt}^0_{\mathcal{A},\mathcal{F}} = 1] - \Pr[\mathsf{Expt}^1_{\mathcal{A},\mathcal{F}} = 1]\right| \leq \delta.$$

Recall that $t_H(L)$ denotes the time to compute the universal hash function H on overall input of length L, that $t(\mathcal{A})$ denotes the running time of algorithm $\mathcal{A}$, and that $q_f(\mathcal{A})$ denotes the number of queries made by $\mathcal{A}$ to its function oracle. We use the following lemma:

Lemma 3.3 *Let F be a family of pseudorandom functions from $\{0,1\}^n$ to $\{0,1\}^n$, and let H be a family of ϵ-XOR universal hash functions from $\{0,1\}^*$ to $\{0,1\}^n$. Define $\mathcal{F}_{K_1,K_2}(N\|M) = F_{K_2}(H_{K_1}(M) \oplus N)$. Then, $\mathcal{F}$ is a family of pseudorandom functions from $\{0,1\}^*$ to $\{0,1\}^n$, and there exists an adversary $\mathcal{A}_1$ such that for every adversary $\mathcal{A}$:*

$$\mathbf{Adv}^{\text{prf}}_{\mathcal{F}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{prf}}_{F}(\mathcal{A}_1) + \epsilon \cdot \binom{q_f(\mathcal{A})}{2}$$

where $t(\mathcal{A}_1) = t(\mathcal{A}) + t_H(L) \cdot q_f(\mathcal{A})$, $q_f(\mathcal{A}_1) = q_f(\mathcal{A})$, and the overall length of message sent by $\mathcal{A}$ to its oracle is L.

Proof: Before beginning the proof, we rewrite the pseudorandom function experiment using our specific scheme:

Experiment $\mathsf{Expt}^b_{\mathcal{A},\mathcal{F}}(1^n)$

1. If $b = 0$ then choose $K_1, K_2 \leftarrow \{0,1\}^n$, and set $\mathcal{O} = F_{K_2} \circ H_{K_1}$.
   Else, if $b = 1$, set $\mathcal{O}$ to be a truly random function $f : \{0,1\}^* \to \{0,1\}^n$.
2. $b' \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(1^n)$
3. Output 1 if and only if $b' = b$

We first change the experiment to $\mathsf{Expt}_{\mathcal{A},f,H}(1^n)$ where K_1 is chosen as above, but a truly random function $f : \{0,1\}^n \to \{0,1\}^n$ is used instead of F_{K_2} in the case of $b = 0$ (and so $\mathcal{O} = f \circ H_{K_1}$ when $b = 0$). A straightforward reduction to the pseudorandomness of F yields that for every adversary $\mathcal{A}_1$,

$$\left|\Pr[\mathsf{Expt}^0_{\mathcal{A},\mathcal{F}} = 1] - \Pr[\mathsf{Expt}_{\mathcal{A},f,H} = 1]\right| \leq \mathbf{Adv}^{\mathrm{prf}}_F(\mathcal{A}_1). \tag{1}$$

The adversary $\mathcal{A}_1$ attacking the underlying pseudorandom function invokes $\mathcal{A}$, chooses K_1 itself and answers every oracle query $M\|N$ of $\mathcal{A}$ by first computing $H_{K_1}(M) \oplus N$ and then sending the result to its oracle. If $\mathcal{A}_1$ received a truly random function as an oracle, then this perfectly simulates $\mathsf{Expt}_{\mathcal{A},f,H}$; in contrast, if $\mathcal{A}_1$ received the pseudorandom function F as its oracle, then this perfectly simulates $\mathsf{Expt}^0_{\mathcal{A},\mathcal{F}}$. The running time of $\mathcal{A}_1$ is exactly that of $\mathcal{A}$ plus q_f computations of H, and the number of queries made by $\mathcal{A}_1$ to its oracle is exactly the same number made by $\mathcal{A}$. Thus, $t(\mathcal{A}_1) = t(\mathcal{A}) + t_H(L) \cdot q_f(\mathcal{A})$, and $q_f(\mathcal{A}_1) = q_f(\mathcal{A})$.

Next, we prove that for every adversary $\mathcal{A}$ making q_f queries to its oracle,

$$\left|\Pr[\mathsf{Expt}_{\mathcal{A},f,H}(1^n) = 1] - \Pr[\mathsf{Expt}^1_{\mathcal{A},\mathcal{F}}(1^n) = 1]\right| \leq \epsilon \cdot \binom{q_f(\mathcal{A})}{2}. \tag{2}$$

In order to see this, consider first the event coll which equals 1 if and only if there exist two queries $M\|N, M'\|N'$ with $M\|N \neq M'\|N'$ made by $\mathcal{A}$ such that $H_{K_1}(M) \oplus N = H_{K_1}(M') \oplus N'$. Then, it holds that

$$\Pr[\mathsf{Expt}_{\mathcal{A},f,H}(1^n) = 1 \mid \neg\mathsf{coll}] = \Pr[\mathsf{Expt}^1_{\mathcal{A},\mathcal{F}}(1^n) = 1].$$

This holds because when $H_{K_1}(M) \oplus N \neq H_{K_1}(M') \oplus N'$ for every pair of distinct $M\|N, M'\|N'$ queried by $\mathcal{A}$ to the oracle, the inputs to f in Expt are all distinct. Thus, the output distribution over $f(H_{K_1}(M) \oplus N)$ in Expt is the same as the output distribution over $f(M\|N)$ in Expt^1. Since

$$\begin{aligned}
&\Pr[\mathsf{Expt}_{\mathcal{A},f,H}(1^n) = 1] \\
&\quad = \Pr[\mathsf{Expt}_{\mathcal{A},f,H}(1^n) = 1 \mid \neg\mathsf{coll}] \cdot \Pr[\neg\mathsf{coll}] \\
&\qquad + \Pr[\mathsf{Expt}_{\mathcal{A},f,H}(1^n) = 1 \mid \mathsf{coll}] \cdot \Pr[\mathsf{coll}] \\
&\quad \leq \Pr[\mathsf{Expt}_{\mathcal{A},f,H}(1^n) = 1 \mid \neg\mathsf{coll}] + \Pr[\mathsf{coll}]
\end{aligned}$$

it remains to prove that

$$\Pr[\mathsf{coll}] \leq \epsilon \cdot \binom{q_f}{2}.$$

In order to see this, observe that $\mathcal{A}$ never receives $H_{K_1}(M) \oplus N$, but rather receives $f(H_{K_1}(M) \oplus N)$ where f is a truly random function. Thus, $\mathcal{A}$ learns nothing about K_1. Intuitively, this means that there will be a collision on the queries made by $\mathcal{A}$ with the same probability that there will be a

collision if all the queries are first made and then K_1 is chosen at random. In order to prove this formally, we modify $\mathsf{Expt}_{\mathcal{A},f,H}(1^n)$ so that in the ith query, the output $f(i)$ is given (we assume without loss of generality that $\mathcal{A}$ never makes the same query twice to the oracle.) Then, at the end of the experiment, K_1 is chosen at random and $H_{K_1}(M) \oplus N$ is computed on all the values $M\|N$ queried to the oracle. As long as no collision takes place, the distribution over the outputs that $\mathcal{A}$ receives is identical in both experiments. Furthermore, if a collision occurs, then it has already occurred and it makes no difference what happens to $\mathcal{A}$'s view afterwards (since a collision already occurred and we are only interested in the question of whether collisions occur). Thus, the collision probability in both experiments is identical.

In this latter experiment, for a series of q_f *distinct* queries $M_1\|N_1, \ldots, M_{q_f}\|N_{q_f}$ to the oracle, we have that

$$\begin{aligned}
\Pr[\mathsf{coll}] &= \Pr\left[\exists i,j \in [q_f] : H_{K_1}(M_i) \oplus N_i = H_{K_1}(M_j) \oplus N_j\right] \\
&= \sum_{i=1}^{q_f-1} \sum_{j=i+1}^{q_f} \Pr[H_{K_1}(M_i) \oplus H_{K_1}(M_j) = N_i \oplus N_j] \\
&= \binom{q_f}{2} \cdot \epsilon.
\end{aligned}$$

where the probability is taken over the choice of K_1 (that specifies the concrete hash function H). Note that the last equality is obtained since H is an ϵ-XOR universal hash function.

Combining Equations (1) and (2), we have that

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{prf}}_{\mathcal{F}}(\mathcal{A}) &= \left|\Pr[\mathsf{Expt}^0_{\mathcal{A},\mathcal{F}} = 1] - \Pr[\mathsf{Expt}^1_{\mathcal{A},\mathcal{F}} = 1]\right| \\
&\leq \left|\Pr[\mathsf{Expt}^0_{\mathcal{A},\mathcal{F}} = 1] - \Pr[\mathsf{Expt}_{\mathcal{A},f,H} = 1]\right| \\
&\quad + \left|\Pr[\mathsf{Expt}_{\mathcal{A},f,H} = 1] - \Pr[\mathsf{Expt}^1_{\mathcal{A},\mathcal{F}} = 1]\right| \\
&\leq \mathbf{Adv}^{\mathrm{prf}}_{\mathcal{F}}(\mathcal{A}_1) + \epsilon \cdot \binom{q_f(\mathcal{A})}{2}
\end{aligned}$$

and this completes the proof. ■

The security of counter mode based on a pseudorandom function is well known. Here we provide the bounds for encryption of messages with at most $2^k - 1$ blocks, and where the initial counter is of length $n - k$ bits. This ensures that as long as the initial counters are all different, then the pseudorandom function is applied to a different input each time. Since we consider the case of random initial counters, it follows that the probability that a counter repeats is at most $\frac{q_E(\mathcal{A})^2}{2^{n-k}}$, where $q_E(\mathcal{A})$ is the number of queries made by adversary $\mathcal{A}$ to the encryption oracle. The reduction to security is very straightforward, with the adversary for the pseudorandom function just querying all the appropriate counters to its oracle. We therefore conclude:

Lemma 3.4 *Let $F : \{0,1\}^n \to \{0,1\}^n$ be a pseudorandom function. Then, there exists an adversary $\mathcal{A}_2$ making such that for every adversary $\mathcal{A}$:*

$$\mathbf{Adv}^{\mathrm{ivE}}_{\mathsf{Enc}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{A}_2) + \frac{q_E(\mathcal{A})^2}{2^{n-k}}.$$

where $t(\mathcal{A}_2) = t(\mathcal{A}) + L_e(\mathcal{A})$ and $q_f(\mathcal{A}_2) = \frac{L_e(\mathcal{A})}{n}$, with q_E being the number of queries made by $\mathcal{A}$ to its encryption oracle, $L_e(\mathcal{A})$ being the total length of all plaintexts queried by $\mathcal{A}$ to its encryption oracle, and q_f being the number of queries made by $\mathcal{A}_2$ to its function oracle.

We are now ready to state the main theorem that provides the security bounds for our construction (we use Π to denote Construction 3.1):

Theorem 3.5 *Let F be a pseudorandom function, and let H be an ϵ-XOR universal hash function. Then, Construction 3.1 is a nonce misuse-resistant authenticated encryption scheme, and there exists an adversary $\mathcal{A}'$ for F such that for every $\mathcal{A}$ attacking Construction 3.1:*

$$\mathbf{Adv}^{\mathrm{mrAE}}_{\Pi}(\mathcal{A}) \leq 2 \cdot \mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{A}') + \epsilon \cdot \binom{q_E(\mathcal{A})}{2} + \frac{q_E(\mathcal{A})^2}{2^{n-k}} + \frac{q_d(\mathcal{A})}{2^n}.$$

(The running time and oracle query complexity of $\mathcal{A}'$ is given in the proof.)

Proof: By Corollary 2.3, we have that:

$$\mathbf{Adv}^{\mathrm{mrAE}}_{\Pi}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{prf}}_{\mathcal{F}}(\mathcal{B}(\mathcal{A})) + \mathbf{Adv}^{\mathrm{ivE}}_{\mathsf{Enc}}(\mathcal{D}(\mathcal{A})) + \frac{q_d(\mathcal{A})}{2^n}$$

where $\mathcal{F}$ is the pseudorandom function used that combines the universal hash and underlying pseudorandom function F. By Lemma 3.3 we have that there exists an adversary $\mathcal{A}_1$ such that

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathcal{F}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{A}_1) + \epsilon \cdot \binom{q_f(\mathcal{A})}{2}$$

and by Lemma 3.4 we have that there exists an adversary $\mathcal{A}_2$ such that

$$\mathbf{Adv}^{\mathrm{ivE}}_{\mathsf{Enc}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{A}_2) + \frac{q_E(\mathcal{A})^2}{2^{n-k}}.$$

Now, adversary $\mathcal{B}(\mathcal{A})$ is a PRF adversary who runs it time at most $t(\mathcal{A}) + 2t_F(L) + t_{\mathsf{Enc}}(L)$ (where L is the total length of the values queried to oracles by $\mathcal{A}$) and asks at most $2(q_E(\mathcal{A})+q_d(\mathcal{A}))$ queries to its oracle, and $\mathcal{A}_1(\mathcal{B}(\mathcal{A}))$ runs in time $t(B(\mathcal{A})) + t_H(L) \cdot q_f(\mathcal{B}(\mathcal{A}))$ and asks $q_f(\mathcal{B}(\mathcal{A}))$ queries to its oracle. Thus, $\mathcal{A}_1$ is an adversary for the pseudorandom function who runs in time $t(\mathcal{A}) + 2t_F(L) + t_{\mathsf{Enc}}(L) + t_H(L) \cdot 2(q_E(\mathcal{A}) + q_d(\mathcal{A}))$ and asks $q_f(\mathcal{A}_1) = 2(q_E(\mathcal{A}) + q_d(\mathcal{A}))$ queries to its oracle.

Furthermore, adversary $\mathcal{D}(\mathcal{A})$ is an ivE encryption adversary who runs in time $t(\mathcal{A}) + t_F(L) + t_{\mathsf{Enc}}(L)$ and asks at most $q_E(\mathcal{A})$ queries to its oracle. Thus, $\mathcal{A}_2$ is an adversary for the pseudorandom function who runs in time $t(\mathcal{A}) + t_F(L) + t_{\mathsf{Enc}}(L) + L_e(\mathcal{A}) \approx t(\mathcal{A}) + t_F(L) + t_{\mathsf{Enc}}(L)$ (we can ignore the $L_e(\mathcal{A})$ factor since it is just the length of the plaintext, whereas $t_{\mathsf{Enc}}(L)$ is the actual cost of encryption which is greater), and asks $q_f(\mathcal{A}_2) = \frac{L_e(\mathcal{A})}{n} < \frac{L}{n}$ queries to its oracle.

Take $\mathcal{A}'$ to be the adversary that incorporates $\mathcal{A}_1$ and $\mathcal{A}_2$. Then, we have that there exists an adversary $\mathcal{A}'$ for the underlying pseudorandom function so that for every adversary $\mathcal{A}$ for the mrAE setting:

- $\mathcal{A}'$ runs in time

$$\begin{aligned}
&[t(\mathcal{A}) + 2t_F(L) + t_{\mathsf{Enc}}(L) + t_H(L) \cdot 2(q_E(\mathcal{A}) + q_d(\mathcal{A}))] \\
&\quad + \ [t(\mathcal{A}) + t_F(L) + t_{\mathsf{Enc}}(L)] \\
&= \ 2 \cdot \Big(t(\mathcal{A}) + t_F(L) + t_{\mathsf{Enc}}(L) + t_H(L) \cdot (q_E(\mathcal{A}) + q_d(\mathcal{A}))\Big)
\end{aligned}$$

  Observe that the running time of $\mathcal{A}'$ is essentially linear in the running time of $\mathcal{A}$ (under the very reasonable assumption that the cost of applying the pseudorandom function to the plaintexts queried by $\mathcal{A}$ and encrypting them, is not more than the running time of $\mathcal{A}$ itself). It is reasonable to therefore writhe that $t(\mathcal{A}') \leq 6 \cdot t(\mathcal{A})$.

- The number of queries made by $\mathcal{A}'$ to its function oracle is at most $2q_E(\mathcal{A}) + 2q_d(\mathcal{A}) + \frac{L}{n}$
- The advantage of $\mathcal{A}$ in the mrAE setting, when reducing to the underlying pseudorandom function F, is

$$\begin{aligned}\mathbf{Adv}^{\mathrm{mrAE}}_{\Pi}(\mathcal{A}) \;\leq\; & \mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{A}') + \mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{A}') \\ & + \epsilon \cdot \binom{q_E(\mathcal{A})}{2} + \frac{q_E(\mathcal{A})^2}{2^{n-k}} + \frac{q_d(\mathcal{A})}{2^n}.\end{aligned} \tag{3}$$

This completes the proof. ■

4 The GCM-SIV Instantiation

In this section, we describe our concrete instantiation of the universal-SIV construction that uses GHASH, which is a part of the GCM specification. Thus, this construction uses the exact same components as GCM in a slightly different way, with the result being nonce misuse resistance. Throughout this section, we use the following lemma, that states the GHASH is indeed a XOR universal hash function:

Lemma 4.1 (Lemma 2 in [17]) *The GHASH function is an ϵ-XOR universal hash function with $\epsilon = \lceil \frac{L}{n} + 1 \rceil \cdot 2^{-t}$, where L is an upper bound on the length of the input, n is the length of the block, and t is the length of the output.*

4.1 Theoretical 3-Key Instantiation

In this instantiation, we simply use the GHASH universal hash function in Construction 3.1. This hash function works by first concatenating zeroes to each of A and M to make them of length that is a multiple of the block length n. Then, an additional block that contains the lengths of both A and M is concatenated (where the length of A is given in the first $n/2$ bits of the block, and the length of M in the last $n/2$ bits). Finally, a polynomial is evaluated over this result.

Observe that GHASH requires a key, the pseudorandom function applied to the output of GHASH requires a key, and finally the pseudorandom function used in counter mode requires a key. Thus, this instantiation requires three separate keys. Although this is a perfectly reasonable instantiation, 3 keys would typically be considered too much for real world usages; we therefore refer to this as a "theoretical instantiation". Later, we present 2-key and 1-key instantiations.

When plugging GHASH directly into Construction 3.1, all that is required is to plug in the value of ϵ given in Lemma 4.1 into the bounds of Theorem 3.5.

Theorem 4.2 (3-Key GCM-SIV) *Construction 3.1 with the pseudorandom function F and the hash function GHASH is a nonce misuse-resistant authenticated encryption scheme. Furthermore, there exists an adversary $\mathcal{A}'$ for F such that for every $\mathcal{A}$ attacking Construction 3.1 making q_E encryption queries and q_d decryption queries of overall length L:*

$$\begin{aligned}&\mathbf{Adv}^{\mathrm{mrAE}}_{\Pi}(\mathcal{A}) \\ &< 2 \cdot \mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{A}') + \frac{\left(\lceil \frac{L}{n} \rceil + 1\right) \cdot q_E(\mathcal{A})^2 + q_d(\mathcal{A})}{2^n} + \frac{q_E(\mathcal{A})^2}{2^{n-k}}\end{aligned}$$

where $t(\mathcal{A}') \leq 6 \cdot t(\mathcal{A})$ and $q_f(\mathcal{A}') \leq 2q_E(\mathcal{A}) + 2q_d(\mathcal{A}) + \frac{L}{n}$.

Proof: We take $t = n$ in Lemma 4.1 and so obtain $\epsilon = \lceil \frac{L}{n} + 1 \rceil \cdot 2^{-n}$. Plugging this into Eq. (3) in the proof of Theorem 3.5, we obtain:

$$\begin{aligned}
&\mathbf{Adv}^{\mathrm{mrAE}}_{\Pi}(\mathcal{A}) \\
&\leq \quad 2 \cdot \mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{A}') + \epsilon \cdot \binom{q_E(\mathcal{A})}{2} + \frac{q_E(\mathcal{A})^2}{2^{n-k}} + \frac{q_d(\mathcal{A})}{2^n} \\
&\leq \quad 2 \cdot \mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{A}') + \frac{\lceil \frac{L}{n} + 1 \rceil}{2^n} \cdot \binom{q_E(\mathcal{A})}{2} + \frac{q_E(\mathcal{A})^2}{2^{n-k}} + \frac{q_d(\mathcal{A})}{2^n} \\
&< \quad 2 \cdot \mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{A}') + \frac{\left(\lceil \frac{L}{n} \rceil + 1\right) \cdot q_E(\mathcal{A})^2 + q_d(\mathcal{A})}{2^n} + \frac{q_E(\mathcal{A})^2}{2^{n-k}} \\
&\leq \quad 2 \cdot \mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{A}') + \frac{2q_E(\mathcal{A})^2}{2^{n-k}} + \frac{q_E(\mathcal{A})^2 + q_d(\mathcal{A})}{2^n}
\end{aligned}$$

where the last inequality holds since $L \leq 2^k \cdot n$ and so $\frac{\lceil L/n \rceil}{2^n} \leq \frac{1}{2^{n-k}}$. The running time and number of oracle queries are taken directly from the proof of Theorem 3.5. ■

In the specific AES instantiation with $n = 128$ and $k = 32$, we conclude:

$$\begin{aligned}
&\mathbf{Adv}^{\mathrm{mrAE}}_{\Pi}(\mathcal{A}) < \\
&2 \cdot \mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{A}') + \frac{q_E(\mathcal{A})^2}{2^{95}} + \frac{q_E(\mathcal{A})^2 + q_d(\mathcal{A})}{2^{128}} \ .
\end{aligned}$$

Note that setting $k = 32$ limits the number of blocks to be 2^{32} and so the maximum length message that can be encrypted is 64GB. This is standard and is used in typical implementations of GCM.

4.2 Two-Key GCM-SIV

In this instantiation, the *same* key is used for the pseudorandom function applied to the output of GHASH and for the counter mode encryption (i.e., we take $K_2 = K_3$). There are two possible ways of doing this. The first is to simply bound the probability that the output of GHASH collides with a possible counter. However, this will result in an additional birthday degradation. The other possibility is to *force* the output from GHASH to always be different from the counters used in the encryption. This is achieved by truncating the output of GHASH to $n - 1$ bits and using an $n - 1$-bit nonce. Then, the most significant bit of the input to the pseudorandom function in order to generate T is *always* zero. Furthermore, the initial counter is taken to be the $n - k - 1$ most significant bits of T followed by k zeroes, and the most significant bit is set to 1. This ensures that the counter never overlaps with the input to the pseudorandom function for generating T. From a security perspective, this means that the same key can be reused with no affect on security at all (a single reduction for the pseudorandom function suffices).

This variant yields the following bounds (obtained as in the 3-key case while changing the exact parameters due to the single bit):

Theorem 4.3 (2-Key GCM-SIV) *Consider the above variant of Construction 3.1 with one key for the pseudorandom function F and one key for the hash function GHASH. Then, the result is a nonce misuse-resistant authenticated encryption scheme, and there exists an adversary $\mathcal{A}'$ for F such that for every $\mathcal{A}$ attacking Construction 3.1 making q_E encryption queries and q_d decryption*

queries of overall length L:[6]

$$\mathbf{Adv}^{\mathrm{mrAE}}_{\Pi}(\mathcal{A}) < 2 \cdot \mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{A}') + \frac{q_E(\mathcal{A})^2}{2^{n-k-2}} + \frac{\left(\lceil \frac{L}{n} \rceil + 1\right) \cdot q_E(\mathcal{A})^2 + q_d(\mathcal{A})}{2^{n-1}} \tag{4}$$

where $t(\mathcal{A}') \leq 6 \cdot t(\mathcal{A})$ *and* $q_f(\mathcal{A}') \leq 2q_E(\mathcal{A}) + 2q_d(\mathcal{A}) + \frac{L}{n}$.

4.3 Single-Key GCM-SIV

In this final instantiation, we take a single key K_0 and derive two keys K_1 and K_2 by computing $K_1 = AES_{K_0}(0^{128})$ and $K_2 = AES_{K_0}(0^{127}\|1)$. We then proceed as in the two-key case. The bounds for here almost identical (with an additional reduction for a *single query* to the pseudorandom function, which is not significant here).

5 The Performance of GCM-SIV

In this section, we provide a detailed performance analysis of our GCM-SIV construction together with experimental results of an optimized software implementation, measured on the latest high end processors with architecture codenames Haswell (HSW) and Broadwell (BDW). The performance of GCM-SIV depends on the message length. We measure the length of the message $|M|$ and the length of the associated data $|A|$ in bytes.

5.1 Encryption Operations

The computational cost of computing GCM-SIV is the following sum:

$$\begin{aligned} &\text{GCM-SIV\_Encryption} \\ &= \text{Key\_Derivation} + \text{GHASH} + \text{Tag\_Generation} \\ &\quad + \text{CTR\_INPUT\_Generation} + \text{CTR\_ENCRYPTION} \end{aligned}$$

We discuss each component separately.

Key_Derivation: Derivation is required only with the one-key GCM-SIV variant (to derive K_1, K_2 from the input key K_0). This derivation requires expanding one AES key and using it to encrypt 2 blocks.

GHASH: This requires a field multiplication in $GF(2^{128})$ for every 16-byte block or part thereof (in the plaintext message M and associated data A), plus a block containing the data length. Thus, the number of field multiplications equals $\ell = \left\lceil \frac{|M|+|A|}{16} \right\rceil + 1$.

Tag_Generation: Generating the authentication tag from the output of GHASH involves executing AES key expansion with K_2, and using it to encrypt a single block (the cost of XORing with IV, and forcing the top bit to 1, is negligible).

CTR_INPUT_Generation: This involves preparing the input blocks to AES in the counter-mode encryption. The cost of forcing the top bit to 0, and incrementing the 32-bit counters (in the least significant quarter of the counter block) is negligible.

[6] In previous versions of this paper, we forgot to add the factor of $\left(\lceil \frac{L}{n} \rceil + 1\right)$ that is multiplied by $q_E(\mathcal{A})^2$ in Eq. (4). We thank Tetsu Iwata and Yannick Seurin for pointing this out.

Full				No Init		
	Cycles			Cycles		
HSW/BDW	GCM-SIV	GCM-SIV	AES-GCM	GCM-SIV	GCM-SIV	AES-GCM
# bytes	Two keys	One key		Two keys	One key	
16	149 / 136	297 / 241	1289 / 1263	133 / 121	133 / 121	178 / 172
32	198 / 171	318 / 284	1277 / 1318	178 / 153	178 / 153	219 / 217
64	322 / 281	444 / 417	1292 / 1335	319 / 278	319 / 278	236 / 238
128	516 / 440	645 / 568	1415 / 1371	282 / 262	282 / 262	293 / 266
256	674 / 566	800 / 694	1558 / 1417	426 / 401	426 / 401	421 / 385
512	966 / 796	1093 / 930	1808 / 1730	722 / 626	722 / 626	760 / 651
1,024	1566 / 1252	1695 / 1385	2312 / 2108	1315 / 1085	1315 / 1085	1252 / 989
1,536	2159 / 1713	2274 / 1843	2816 / 2416	1907 / 1544	1907 / 1544	1714 / 1305
2,048	2751 / 2171	2869 / 2300	3372 / 2842	2498 / 1996	2498 / 1996	2287 / 1765
4,096	5118 / 4005	5244 / 4136	5332 / 4354	4867 / 3837	4867 / 3837	4296 / 3243
8,192	9862 / 7666	9994 / 7782	9521 / 7388	9611 / 7498	9611 / 7498	8399 / 6289
	C/B			C/B		
8,192	1.2/0.94	1.22/0.95	1.16/0.9	1.17/0.92	1.17/0.92	1.03/0.77

Table 2: GCM-SIV encryption performance for different message lengths, on the Haswell and Broadwell (HSW/BDW) architectures. Comparison to the performance of AES-GCM (OpenSSL 1.0.2) is provided. The numbers are in cycles, except for the last row which reports the performance in C/B.

CTR_ENCRYPTION This is the cost of $\left\lceil \frac{|M|}{16} \right\rceil$ AES operations on preprepared input. (Note that the key used has already been expanded when preparing the tag.)

5.2 Implementation Optimizations

Software implementations on high end Intel processors use the AES-NI and the PCLMULQDQ instructions. There exist optimizations that improve the performance significantly, compared to straightforward implementations. We briefly describe these optimizations.

1. The key derivation (required only for one-key GCM-SIV) can be reduced by a new software optimization that pipelines the instructions efficiently. We were able to execute this derivation in 84 cycles (on HSW/BDW architectures).

2. When the message includes more than 8 blocks, then $GHASH$ can be optimized by: **(1)** Preparing a lookup table with "powers" of H, **(2)** Interleaving the polynomial multiplications, and **(3)** Deferring the reduction modulo $Q(x)$ (the field polynomial) to take place only once per 8 blocks [10, 11]. See also [8]for an analysis and an improved reduction method. Effectively, this reduces the cost of $GHASH$ to ℓ polynomial multiplications + $\frac{1}{8}\ell$ reductions, instead of ℓ field multiplications. We were able to compute $GHASH$ using this method at the asymptotic performance of 0.56 and 0.3 cycles per byte (C/B) on HSW and BDW, respectively (for an 8KB message).

3. For long enough messages, the encryption can operate on 8 blocks in parallel, interleaving AESENC/AESENCLAST instructions [7, 9]. We were able to encrypt at the asymptotic performance of 0.63 C/B on both HSW and BDW.

Remark 5.1 *For long messages the setup cost is small, and the differences between one key and two keys become negligible. From the above data, we can predict the performance (for long messages) to be the sum of GHASH and encryption, which is* 1.19 *C/B for HSW, and* 0.93 *C/B for BDW.*

5.3 Theoretical Comparison to AES-GCM

AES-GCM uses CTR mode for encryption, and *GHASH* for the authentication. It uses a single key, and involves a derivation step: $H = AES_K(0^{128})$ and $MASK = AES_K(\cdot)$.[7]

When counting the *number of operations*, GCM and GCM-SIV have roughly the same performance. In fact, GCM-SIV is slightly cheaper due to a simpler counter incrementing. However, for encryption, the main difference between GCM and GCM-SIV is in the possible order of operations. By definition, GCM-SIV can start AES-CTR encryption only after the authentication tag has been computed. By contrast, GCM can interleave the AES and GHASH computations (for the message; not for the AAD). This enables GCM encryption to be faster than GCM-SIV.

Note, however, that for decryption GCM-SIV can also interleave the AES and GHASH computations, and so its performance is *identical* to that of GCM.

5.4 Experimental Results

For our study, we prepared an optimized software implementation of GCM-SIV, and measured it on the Haswell and Broadwell (HSW/BDW) architectures. The results are summarized in Table 2. The table provides the cycles count for GCM-SIV for various message lengths, to illustrate the performance characteristics. They are compared to the optimized AES-GCM implementation of OpenSSL (1.0.2). We note that the cost of the "Init" step in OpenSSL is approximately 1,100 cycles. This includes, among other operations the preparation of a lookup table, keys setup, and more. Therefore, to facilitate a more detailed comparison, Table 1 also shows the AES-GCM performance without the Init step, as well as GCM-SIV without the initialization (this neutralises the fact that OpenSSL carries out more operations in its Init than we do in our implementation). Needless to say, the two-key and one-key variants are identical after Init, as can be seen in the table.

The last row of the table shows the performance in C/B, for a long message. Note that the measured performance matches the predictions of Remark 5.1.

The methodology used for carrying out these measurements is as follows (and is the same for GCM-SIV and AES-GCM). The following process was repeated 30 times: compute the operation 500 times for a "warmup" (e.g., to place code/data in the caches). Then, compute and clock the operation 500 times, and take the average result. The output appearing in the table is the minimum value over the 30 runs. The reason that we take the minimum is to neutralize noise caused by interrupts to the operating system.

All the runs were carried out on a system where the Intel® Turbo Boost Technology, the Intel® Hyper-Threading Technology, and the Enhanced Intel Speedstep® Technology, were *disabled*.

The results show that up to 32 bytes (including), GCM-SIV with 2 keys is faster even than GCM without Init. Therefore, for key wrap, GCM-SIV is an excellent choice. It is also very efficient for scenarios that encrypt many short messages with the same key (since the key derivation is carried out only once here and so the cost is like without Init).

[7] The mask is XOR-ed with the GHASH result, to make it a MAC tag. Here, $\cdot$ denotes the first counter block used in AES-GCM.

For long messages, as expected, we see only a very small difference between the 2-key and 1-key versions of GCM-SIV, in the "full" implementation. This allows for choosing the more cost effective variant (i.e., 1 key) from the network traffic viewpoint. *We observe that on the latest Broadwell architecture, the cost of GCM-SIV encryption falls below* 1 *cycle per byte.*

For encryption, Table 2 shows that GCM-SIV is 14% slower than AES-GCM on Haswell, and 19% slower than AES-GCM on Broadwell. The reason for this difference is that the optimized AES-GCM software is able to interleave AES and $GHASH$ computations, while GCM-SIV cannot. Recall that nonce misuse-resistance provably requires two passes, and thus there is an inevitable cost incurred. However, we point out that for decryption, optimized AES-GCM and GCM-SIV would have the same performance because the AES and $GHASH$ operations can be interleaved.

We comment about the performance of GCM-SIV *without initialization* for 64 and 128 bytes messages. Our optimized GHASH code prepares a lookup table to aggregate 8 block multiplications before the reduction step. Of course, this becomes relevant only when the message length is at least 128 bytes. If the cost of the setup is not (including the preparation of the table), then this leads to the seeming anomaly in Table 2 where 128-byte GCM-SIV takes less time than 64-byte GCM-SIV.

References

[1] F. Abed, C. Forler and S. Lucks. Classification of the CAESAR Candidates. *Cryptology ePrint Archive*, 2014/792. `http://eprint.iacr.org/2014/792.pdf`.

[2] L. Bahack. Julius. `http://competitions. cr.yp.to/caesar-submissions.html`, 2014.

[3] A. Bogdanov, M.M. Lauridsen and E. Tischhauser. AES-Based Authenticated Encryption Modes in Parallel High-Performance Software. *IACR Cryptology ePrint Archive*, report 2014:186, 2014.

[4] E. Andreeva, A. Bogdanov, A. Luykx, B. Mennink, E. Tischhauser and K. Yasuda. AES-COPA v.1. *CAESAR competition submission.*

[5] M. Bellare, and C. Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In *ASIACRYPT 2000*, Springer (LNCS 1976), pages 531–545, 2000.

[6] E. Fleischmann, C. Forler and S. Lucks. McOE: A Family of Almost Foolproof On-Line Authenticated Encryption Schemes. In *FSE 2012*, Springer (LNCS 7549), pages 196–215, 2012.

[7] S. Gueron. Intel Advanced Encryption Standard (AES) Instructions Set, Rev 3.01. Intel Software Network. (2012) `https://software.intel.com/en-us/articles/intel-advanced-encryption-standard-aes-instructions-set`

[8] S. Gueron, AES-GCM for Efficient Authenticated Encryption – Ending the Reign of HMAC-SHA-1? In *Real-World Cryptography*, 2013. `https://crypto.stanford.edu/RealWorldCrypto/ slides/gueron.pdf`

[9] S. Gueron. Intel's New AES Instructions for Enhanced Performance and Security. 16*th FSE* (FSE 2009), Springer (LNCS 5665), pages 51–66, 2009.

[10] S. Gueron, M. E. Kounavis. Intel Carry-Less Multiplication and Its Usage for Computing The GCM Mode, Rev 2.01. Intel Software Network. http://software.intel.com/sites/default/files/article/165685/clmul-wp-rev-2.01-2012-09-21.pdf

[11] S. Gueron, M. E. Kounavis. Efficient Implementation of the Galois Counter Mode Using a Carry-less Multiplier and a Fast Reduction Algorithm. *Information Processing Letters* 110:549–553, 2010.

[12] V.T. Hoang, T. Krovetz and P. Rogaway. Robust Authenticated-Encryption: AEZ and the Problem That It Solves. In *EUROCRYPT 2015*, Springer (LNCS 9056), pages 15–44, 2015.

[13] H. Krawczyk. The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?). In *CRYPTO 2001*, Springer (LNCS 2139), pages 310–331, 2001.

[14] T. Iwata and K. Yasuda. HBS: A Single-Key Mode of Operation for Deterministic Authenticated Encryption. In *FSE 2009*, Springer (LNCS 5665), pages 394–415, 2009.

[15] T. Iwata and K. Yasuda. BTM: A Single-Key, Inverse-Cipher-Free Mode for Deterministic Authenticated Encryption. In *Selected Areas in Cryptography 2009*, Springer (LNCS 5867), pages 313–330, 2009.

[16] T. Krovetz. HS1-SIV. http://competitions. cr.yp.to/caesar-submissions.html, 2014.

[17] 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.

[18] C. Namprempre, P. Rogaway and T. Shrimpton. Reconsidering Generic Composition. In *EUROCRYPT 2014*.

[19] R. Reyhanitabar, S. Vaudenay and D. Vizár. Misuse-Resistant Variants of the OMD Authenticated Encryption Mode. In *ProvSec 2014*, Springer (LNCS 8782), pages 55–70, 2014.

[20] 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.

A Definitions

We use the following definitions taken from [20, 18]. We begin by defining CPA-secure IV-based encryption and nonce-based authenticated-encryption. The difference between IV-based and nonce-based encryption is that in the former a random IV is used, whereas in the latter a unique nonce is provided as input in every encryption. The guarantee of nonce-based encryption is security is maintained as long as the nonce used is different each time. Beyond that, we consider CPA security for IV-based encryption and authenticated encryption for nonce-based encryption. We do this since this is what we need for our construction. We adopt the definition of authenticated encryption from [20, 18] who replace the encryption oracle with a random function, and mandate that the decryption oracle always outputs $\perp$. Observe that this means that the adversary – who is

given oracle access to either the pair of oracles $\mathsf{Enc}_K, \mathsf{Dec}_K$ or a random function and $\perp$ – is not allowed to query the output of an "encryption" query to its "decryption" oracle. This is because when given $\mathsf{Enc}_K, \mathsf{Dec}_K$ the decryption query will return the plaintext queries to Enc_K, whereas when given a random function and $\perp$ the decryption query will return $\perp$. Thus, it will be trivial to distinguish. Nevertheless, this restriction is without loss of generality since any adversary who makes such queries can be converted to an adversary who does not make such queries and succeeds with exactly the same probability (if such a query is asked, just return the plaintext it was generated from).

CPA-secure IV-based encryption (ivE). Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an IV-based encryption scheme, and let the space of IVs be $\mathcal{IV}$. Encryption involves choosing $IV \leftarrow \mathcal{IV}$ uniformly at random, and then computing the deterministic function $\mathsf{Enc}_K(IV, M)$. Consider the following oracles:

- *Oracle* Enc_K*:* upon input $M \in \{0,1\}^*$, it chooses $IV \leftarrow \mathcal{IV}$ at random and computes $C = \mathsf{Enc}_K(IV, M)$. The output is $IV \| C$.
- *Oracle* $\$_K$*:* upon input $M \in \{0,1\}^*$, it chooses $IV \leftarrow \mathcal{IV}$ at random and computes $C = \mathsf{Enc}_K(IV, M)$. The output is a random string of length $|IV \| C|$.

The advantage of an adversary $\mathcal{A}$ against an IV-based encryption scheme is defined to be:

$$\mathbf{Adv}^{\mathrm{ivE}}_{\Pi}(\mathcal{A}) = \left| \Pr_K \left[\mathcal{A}^{\mathsf{Enc}_K(\cdot)} = 1 \right] - \Pr_K \left[\mathcal{A}^{\$_K(\cdot)} = 1 \right] \right|$$

We say that Π is a CPA-secure IV-based encryption scheme if for every probabilistic-polynomial time adversary $\mathcal{A}$ there exists a negligible function μ such that $\mathbf{Adv}^{\mathrm{ivE}}_{\Pi}(\mathcal{A}) \leq \mu(n)$.

Secure nonce-based authenticated encryption (nAE). Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a nonce-based encryption scheme. Encryption is a deterministic function receiving a key K, nonce N, associated data A and plaintext message M, and is denoted $C = \mathsf{Enc}_K(N, A, M)$. We denote decryption by $\mathsf{Dec}_K(N, A, C)$. Consider the following oracles:

- *Oracle* $\$_K$*:* upon input (N, A, M), it computes $C = \mathsf{Enc}_K(N, A, M)$. If $C = \perp$ then the output is $\perp$; otherwise, the output is a random string of length $|C|$.
- *Oracle* $\perp$*:* upon any input, returns $\perp$.

The advantage of an adversary $\mathcal{A}$ against a nonce-based authenticated encryption scheme is defined to be:

$$\begin{aligned} \mathbf{Adv}^{\mathrm{nAE}}_{\Pi}(\mathcal{A}) = \Big| & \Pr_K \left[\mathcal{A}^{\mathsf{Enc}_K(\cdot,\cdot,\cdot), \mathsf{Dec}_K(\cdot,\cdot,\cdot)} = 1 \right] \\ & - \Pr_K \left[\mathcal{A}^{\$_K(\cdot,\cdot,\cdot), \perp(\cdot,\cdot,\cdot)} = 1 \right] \Big| \end{aligned}$$

where $\mathcal{A}$ may not make two queries $(N, M, C), (N, M', C')$ to Enc with the same first component (nonce), and may not make any decryption query for a value (N, A, C) that was obtained as output from some query to Enc. We say that Π is a secure nonce-based authenticated encryption scheme if for every probabilistic-polynomial time adversary $\mathcal{A}$ there exists a negligible function μ such that $\mathbf{Adv}^{\mathrm{nAE}}_{\Pi}(\mathcal{A}) \leq \mu(n)$.

Secure nonce misuse-resistant authenticated encryption (mrAE). We define a notion of misuse resistance for *nonce-based* encryption. Specifically, we define nonce misuse-resistance where full security is guaranteed to hold as long as the same nonce is not used for the same message and associated data. Since nonce-based encryption is deterministic, when the same nonce *is* used for the same message and associated data, the same ciphertext is obtained. Thus, the only information revealed to the adversary is that the same message was encrypted. It is immediate that nonce-based misuse-resistant encryption implies IV misuse-resistant encryption by simply using a random IV (and noting that for q encryptions the probability that an IV repeats is $q^2/2^\ell$ where ℓ is length of the IV).

Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a nonce-based encryption scheme. Encryption is a deterministic function receiving a key K, nonce N, associated data A and plaintext message M, and is denoted $C = \mathsf{Enc}_K(N, A, M)$. We denote decryption by $\mathsf{Dec}_K(N, A, C)$. Consider the following oracles:

- *Oracle* $\$_K$*:* upon input (N, A, M), it computes $C = \mathsf{Enc}_K(N, A, M)$. If $C = \perp$ then the output is $\perp$; otherwise, the output is a random string of length $|C|$.
- *Oracle* $\perp$*:* upon any input, returns $\perp$.

The advantage of an adversary $\mathcal{A}$ against a nonce-based authenticated encryption scheme is defined to be:

$$\mathbf{Adv}^{\mathrm{mrAE}}_{\Pi}(\mathcal{A}) = \Big|\mathrm{Pr}_K\left[\mathcal{A}^{\mathsf{Enc}_K(\cdot,\cdot,\cdot),\mathsf{Dec}_K(\cdot,\cdot,\cdot)} = 1\right] - \mathrm{Pr}_K\left[\mathcal{A}^{\$_K(\cdot,\cdot,\cdot),\perp(\cdot,\cdot,\cdot)} = 1\right]\Big|$$

where $\mathcal{A}$ may not make two identical queries to Enc (with the same nonce, associated data and message), and may not make any decryption query for a value (N, A, C) that was obtained as output from some query to Enc. We say that Π is a secure nonce misuse-resistant authenticated encryption scheme if for every probabilistic-polynomial time adversary $\mathcal{A}$ there exists a negligible function μ such that $\mathbf{Adv}^{\mathrm{nAE}}_{\Pi}(\mathcal{A}) \leq \mu(n)$.

Nonce misuse-resistant authenticated encryption is the same as nonce-based authenticated encryption with the except that if a nonce is reused then the only damage is that an adversary can know if the AD and plaintext message in two messages with the same nonce are the same or different, but nothing more. This is because Enc is deterministic and so a nonce reused with exactly the same associated data and plaintext message will give the same ciphertext, whereas by the definition above whenever either the nonce or the associated data or the plaintext message is different, the result is completely different (and indistinguishable from random). We do not allow $\mathcal{A}$ to make two identical queries since the \$ oracle always returns a fresh random string and so this is a trivial (but meaningless) distinguisher.