

Flexible and Efficient Message Authentication in Hardware and Software

David A. McGrew[†] and John Viega[‡]

[†] Cisco Systems, Inc., [‡] Secure Software

Abstract. We present the Galois Message Authentication Code (GMAC), a generic construction based on universal hashing using multiplication in the finite field $GF(2^{128})$. We also present GCM, a block cipher mode of operation that provides both encryption and message integrity in a single primitive, and is based on GMAC. The inherent parallelism in our constructs enable hardware implementations to achieve speeds greater than 10 gigabits per second, while requiring significantly less area than other constructions. Software implementations also have excellent performance characteristics.

GMAC accepts nonces of arbitrary length using a keyed PRF based on the underlying hash function, GHASH. Additionally, we demonstrate how our MAC construction can be used incrementally and we provide a proof of security for our constructs to satisfying bounds under the PRP assumption.

Both GMAC and GCM are, to the best of our knowledge, free of intellectual property restrictions.

Keywords: Authenticated encryption with associated data, parallelism

1 Introduction

There is a compelling need for a new message authentication code (MAC) that works at speeds of 10 gigabits per second and higher in hardware, performs acceptably in software and is free of intellectual property restrictions. For example, the IEEE linksec group needs to decide on a standard message integrity scheme that satisfies these goals.

Most applications that require message privacy also require message authentication. Combining these two requirements securely is recognized as a challenging task that is rarely done securely using *ad hoc* constructions. As a result, there are several recent proposals for dedicated block cipher modes of operation that provide authenticated encryption. Most such schemes are *authenticated encryption with associated data (AEAD)* schemes, meaning that they are capable of authenticating more data than they encrypt.

Until recently, the leading authenticated encryption scheme was OCB [12], which builds upon Jutla's work in this area [8], and is similar to several constructs from Gligor and Donescu [7]. Note that OCB is not an AEAD scheme, though

the authors have recommended a way to use OCB that accepts associated plaintext data. OCB-AES was under consideration for standardization by the IEEE 802.11i committee, but the intellectual property issues, along with the presence of a compelling alternative made it unacceptable as a mandatory-to-implement algorithm.

CCM was created in response to intellectual property concerns with OCB [16]. CCM is an AEAD scheme that combines CTR mode encryption and CBC-MAC. This mode was eventually selected as the required algorithm for 802.11i and is now a NIST-approved mode of operation for AES.

Despite the success of CCM, there are two inherent limitations that make it unacceptable for new applications. First, it requires the length of a message to be known in advance. This can be a problem for handling, for example, IPv6 jumbograms in a hardware device, since the device would require a 4GB buffer. More importantly, the message authentication portion of CCM is neither parallelizable nor pipelinable, making cost-effective hardware scalability to 10 gigabit speeds highly impractical.

The EAX encryption mode [4] is an authenticated encryption scheme that is similar to CCM in that it is a combination of CTR mode and CBC-MAC. However, it is simpler in its specification and proof of security, removes the limitation of requiring the message length up-front and introduces the ability to have nonces of arbitrary length, which is appealing since it removes the burden of allocating scarce bits in a small nonce space. Unfortunately, EAX still uses CBC-MAC as a component, and as a result scales no better for high-speed applications.

The recently proposed CWC mode of operation [9] is an AEAD scheme that is parallelizable and is thus able to scale to meet the needs of high-speed networking. Additionally, CWC is free of intellectual property restrictions. However, CWC requires use of large-integer modular multiplication operations that consume considerable circuit area in hardware designs and are impractical for platforms without full 32-bit multipliers. Plus, the block size of the message authentication portion is not aligned with common block sizes for block ciphers, adding complexity to buffering algorithms for real implementations. The cost of CWC in both hardware and software is acknowledged to be greater than OCB (though the software speed differences are probably not significant enough to matter for practical applications). The authors of the CWC algorithm have publicly stated that there would be no compelling reason to use the mode of operation were OCB free of intellectual property restrictions.

In this paper, we introduce GCM (Galois and Counter Mode), an AEAD scheme that, like CWC and OCB, scales to high speeds in hardware. This mode is compelling in that it requires less circuit area in hardware implementations than other alternatives, performs competitively in software implementations and is free of intellectual property restrictions. Additionally, it allows for an arbitrary-length nonce, which CWC does not do. GCM is based on GMAC, a new Carter-Wegmen style MAC [15] using multiplication over the binary finite field $GF(2^{128})$. In Section 2 we introduce GMAC, and then in Section 3 define

GPRF, the underlying keyed pseudo-random function used as a component in our schemes. In Section 5 we discuss details of math in $GF(2^{128})$, including implementations in both hardware and software. Then, in section 4, we introduce GCM mode, which is built on top of GMAC and GPRF.

There is an additional need in storage area networks (SANs) for message authentication that can efficiently authenticate large, remote storage. Such applications have requirements well-met by incremental message authentication [2], meaning that a small change in data requires only a small (proportional) amount of work in order to “update” the message authentication value. To date, the only recognized incremental message authentication construction is XOR MAC [3], which is subject to intellectual property restrictions. We discuss our scheme for incremental message authentication in Section 6.

2 A MAC using universal hashing over $GF(2^{128})$

The two main operations used in the algorithm are block cipher encryption and multiplication over the field $GF(2^{128})$. At a high level, it consists of hashing the data with the function GHASH and then postprocessing the final result by encrypting a nonce using a block cipher, and XORing the two values together. The function GHASH multiplies each 128-bit block of the data by a 128-bit secret key K using multiplication over $GF(2^{128})$ and accumulates the result. Before defining our functions, we summarize our notation. We denote the multiplication of two elements $X, Y \in GF(2^{128})$ as $X \cdot Y$, and denote the addition of X and Y as $X \oplus Y$ (addition in $GF(2^{128})$ is equivalent to XOR, as discussed in Section 5). The function $\text{len}()$ returns a 64-bit nonnegative integer containing number of bits in its argument. When converting between elements of $GF(2^{128})$, bit strings, and integers, we use the convention that the first bit in the string, the most significant bit, and highest polynomial coefficient are all on the left. We use the convention that the first bit in a string has the index zero. The expression 0^l denotes a string of l zero bits, and $A||B$ denotes the concatenation of two bit strings A and B . The function $\text{trunc}()$ returns only the leftmost t bits of its argument.

GMAC is defined in Algorithm 1. It authenticates two distinct bit strings, which facilitates its use in a mode of operation that provides both confidentiality and message authentication. We denote these inputs as A and C . Below we identify A as authenticated-only data and C as confidential-and-authenticated data. The other inputs to GMAC are a secret hash key $(K_b, K_n, K_h) \in (\{0, 1\}^{128})^3$ and a variable-length nonce N . For each fixed value of the key, each value of the nonce must be distinct, or must be chosen at random with a negligible probability of not being distinct. GMAC returns a t -bit authentication tag, where $t \leq 128$. Algorithm 1 defines GMAC in terms of a pseudorandom function $\text{GPRF} : \{0, 1\}^{128} \times \{0, 1\}^{128} \times \{0, 1\}^l \rightarrow \{0, 1\}^m$, which takes two 128-bit keys and an l -bit nonce and returns m bits. This function is defined below. GMAC is secure when GPRF is secure, as is formalized in Theorem 2. Algorithm 1 specifies an authentication tag generation algorithm. The corresponding message/tag

verification algorithm consists of computing the MAC value of a message using the secret key, and accepting message/tag pairs only when the tag matches the computed value.

Algorithm 1 Algorithm GMAC, which takes as its inputs two bit strings A and C , a secret key (K_b, K_n, K_h) , and a nonce N , and returns a t -bit message authentication code.

```

 $X \leftarrow 0$ 
append zero bits to  $A$  until  $\text{len}(A)$  is a multiple of 128.
for  $i$  from 0 to  $\text{len}(A)/128 - 1$  do
    set  $M$  to bits  $128i$  through  $128i + 127$  of  $A$ .
     $X \leftarrow (X \oplus M) \cdot K_h$ 
end for
append zero bits to  $C$  until  $\text{len}(C)$  is a multiple of 128.
for  $i$  from 0 to  $\text{len}(C)/128 - 1$  do
    set  $M$  to bits  $128i$  through  $128i + 127$  of  $C$ .
     $X \leftarrow (X \oplus M) \cdot K_h$ 
end for
 $X \leftarrow (X \oplus (\text{len}(A) \parallel \text{len}(C))) \cdot K_h$ .
set  $S$  to the leftmost  $t$  bits of  $\text{GPRF}(K_n, K_b, N)$ .
return the leftmost  $t$  bits of  $X \oplus S$ 

```

The algorithm GHASH is identical to GMAC without the final steps of generating a GPRF output and adding it into the hash value. In other words, GHASH runs through Algorithm 1 then returns the leftmost t bits of X . The ‘multiply and accumulate’ aspect of this algorithm is equivalent to the use of Horner’s rule to evaluate a polynomial in K over $GF(2^{128})$. This method computes the polynomial $\bigoplus_{i=1,n} M_i \cdot K^i$ using the operations

$$((\dots(((M_n K) \oplus M_{n-1})K \oplus M_{n-2})K \dots M_1)K) \oplus M_0 \quad (1)$$

In order to illustrate this correspondence, and to establish ideas used in the proof of security, we re-express Algorithm 1 in algebraic terms. Each input to our algorithm is a bit string, but our algorithms work on elements of $GF(2^{128})$. We use the following convention for mapping a bit string M onto a sequence of elements in $GF(2^{128})$. The bit string is decatenated into 128-bit blocks, padding the final block with zero bits if needed. The leftmost block, which contains bits zero through 127, is identified as the first element M_1 , and the rightmost block is identified as the n^{th} element M_n . Symbolically,

$$M_1 \parallel M_2 \parallel \dots \parallel M_{n-1} \parallel M_n = M \parallel 0^r \quad (2)$$

where $M_i \in GF(2^{128})$ for $1 \leq i \leq n$, and r is the number of bits of padding appended to M in order to make $\text{len}(M)$ a multiple of 128. GMAC can be

expressed algebraically as

$$\text{GMAC}(K_h, K_b, N, A, C) = \text{trunc}(((\text{len}(A)\|\text{len}(C)) \oplus \bigoplus_{i=1,n} M_i \cdot K_h^{n-i+1}) \cdot K \oplus \text{GPRF}(K_b, N)) \quad (3)$$

Note that the powers of K_h appear in the reverse order than the coefficients in order to match the natural ordering of the elements (as in Equation 2).

3 GPRF

We use our universal hash function to make GPRF, a keyed pseudorandom function with a variable input length as well as a variable output length. We define the function in such a way that it produces a conceptually infinite output that should be truncated to the desired length. The function’s ability to accept inputs larger than the cipher’s block size relieves its user of the burden of apportioning a fixed-sized nonce. This is important in practice, because many applications have data types that one would like to use as nonces, but which are large and which may have a variable length. Two examples of data that one might want to include in nonces are Internet Protocol Version 6 (IPv6) addresses, which are 128 bits long, and fully qualified domain names (FQDNs), which have variable lengths and are usually longer than the AES block size. This primitive can be useful in applications as diverse as key derivation, encryption, and message authentication.

GPRF uses a pseudorandom permutation $E : \{0, 1\}^{128} \rightarrow \{0, 1\}^{128}$ in counter mode and a universal hash function over $GF(2^{128})$. In principle, any 128-bit block cipher could be used, but we specify AES with a 128-bit key. We use a counter mode definition that is based on that actually used in practice, in which the 128-bit counter $C = F\|I$ is divided into a 32-bit incrementing part I and a 96-bit fixed part F . Successive counters are generated using the function $\mathbf{incr}()$, which sets I to $I+1 \bmod 2^{32}$ and leaves F unchanged¹. This counter definition matches that used in several current and proposed standards. We denote as $S_i(N)$ the i^{th} block of output generated using the nonce N . GPRF is defined by the equations

$$\begin{aligned} \text{GPRF}(K_n, K_b, N) &= S_0(N)\|S_1(N)\|\dots \\ S_i(N) &= E(K_b, C_i(N)) \\ C_i(N) &= \begin{cases} H(K_n, N) & \text{if } i = 0, \\ \mathbf{incr}(C_{i-1}(N)) & \text{otherwise.} \end{cases} \end{aligned}$$

where the hash function H is defined as

$$H(K, N) = \begin{cases} N\|0^{32} & \text{if } \text{len}(N) = 96, \\ K \cdot ((0^{64}\|\text{len}(N)) \oplus \bigoplus_{i=1,n} N_i \cdot K^{n-i+1}) & \text{otherwise.} \end{cases} \quad (4)$$

¹ This counter format avoids the need to implement a 128-bit increment in hardware (which has a high circuit complexity at high speeds) or an LFSR in software (which would take about eight times as many clock cycles on a 32-bit CPU).

The definition of H contains an optimization for applications that use small, fixed size nonces: whenever a nonce is exactly 96 bits long, we just return the nonce itself rather than apply the hash to it. Perhaps surprisingly, this modification does not degrade the efficacy of the hash function, as we show in Theorem 1.

We include this optimization because the flexibility provided by variable-size nonces does not provide any additional value to applications for which the nonces have small, fixed sizes. In these cases, implementers may quite reasonably wish to avoid the extra step of hashing the nonce and the extra storage requirement of the key used in that step. We expect that implementations of GPRF that are intended for applications where nonces are small and have a fixed size will not implement the nonce-hashing component of the function. The IPsec ESP counter mode standard is one such case. However, the fact that GPRF can accept longer nonces offers flexibility to future versions of the standard, and provides interoperability between implementations that need the flexibility and those that choose to ignore it.

4 GCM: a combined mode

GCM is a simple combination of GMAC with CTR mode using the encrypt-then-authenticate paradigm, which has been shown to be secure way to combine encryption and integrity schemes in the generic case [10]. GCM encryption is defined in Algorithm 2 and decryption is defined in Algorithm 3. Note that GPRF is used in two ways: directly for encryption, and indirectly for authentication. Importantly, the first 128 bits of the output of GPRF is used only for authentication, and the remaining output is used only for encryption. This mode provides integrity validation for both the associated data A and the plaintext message P , while only providing confidentiality for P . We assume that the keys K_b, K_n and K_h are selected uniformly at random, though some implementations may derive all of these keys from a single base key K by using K to encrypt fixed values. Additionally, we speculate that there are ways to truly collapse these into one or two keys with only minor changes to GPRF. However, validating or refuting this is left as an interesting area of future research.

Algorithm 2 Algorithm GCM-Encrypt, which takes as its input two bit strings A and P , a secret key (K_b, K_n, K_h) , and a nonce N , and returns a $\text{len}(P) + t$ -bit result (where t is the selected output length used with GMAC).

```

set  $S$  to bits 128 through  $128 + \text{len}(P)$  of  $\text{GPRF}(K_n, K_b, N)$ 
 $C \leftarrow P \oplus S$ 
 $T \leftarrow \text{GMAC}(A, C, K_h, N)$ 
return  $C||T$ 

```

The security of GCM is captured in Theorems 1 and 2. The proofs for these theorems appear in Appendix A. We use indistinguishability from random as our

Algorithm 3 Algorithm GCM-Decrypt, which takes as its input two bit strings A and $C||T$, a secret key (K_b, K_n, K_h) , and a nonce N , and either returns failure or a result of size $\text{len}(C)$.

```

 $T' \leftarrow \text{GMAC}(A, C, K_h, N)$ 
if  $T \neq T'$  then
    return FAIL
end if
set  $S$  to bits 128 through  $128 + \text{len}(C)$  of  $\text{GPRF}(K_n, K_b, N)$ 
return  $C \oplus S$ 

```

model for the security of a keyed PRF. This model is based on the experiment in which an adversary is given access to an oracle that returns an output when given an input. Inside the ‘back box’ that implements the oracle is either a keyed PRF with a randomly selected secret key or a truly random function.; the adversary is challenged to tell the difference. Before the experiment, a bit b that determines if the oracle is random or is a PRF is chosen uniformly at random. After the experiment, the adversary returns a bit d indicating her guess as to which function is in the black box. We use the following conventions:

$$\begin{aligned}
 b &= 1 \text{ if the oracle is a PRF with a randomly selected key,} & (5) \\
 b &= 0 \text{ if the oracle is a true random permutation,} \\
 d &= 1 \text{ if the adversary guesses that } b = 1, \\
 d &= 0 \text{ if the adversary guesses that } b = 0.
 \end{aligned}$$

The *distinguishing advantage* A_d of the adversary is defined as the adversary’s true positive probability less her false positive probability, that is,

$$A_d = \mathbf{P}[d = 1 \mid b = 1] - \mathbf{P}[d = 1 \mid b = 0]. \quad (6)$$

If the adversary can distinguish between the two cases with non-negligible probability, then the advantage is non-negligible, and vice-versa. In the experiment, we restrict the number of queries that the adversary can make to the oracle to some value q .

Theorem 1. *If there is an adversary that can distinguish GPRF from a random function with advantage A_{GPRF} , when the output of that function is limited to q invocations with a maximum output length of l bits per invocation, then that adversary can distinguish E from a PRP with advantage A_{PRP} where*

$$\begin{aligned}
 A_{PRP} &\geq A_{GPRF} - q^2 (\lceil l/128 \rceil^2 2^{-128} + \lceil l/128 + 1 \rceil^3 2^{-126}) \\
 &\simeq A_{GPRF} - q^2 l^2 2^{-142} + q^2 l^3 2^{-147}.
 \end{aligned} \quad (7)$$

We use the standard model for the security of a MAC in the presence of a chosen-message attack, in which an adversary is given access to a tag generation oracle and a message/tag verification oracle. The adversary can pass messages to the tag generation oracle one at a time, then construct any message/tag pairs

that it likes and send these to the verification oracle. The forgery advantage A_f is the probability that the adversary can get the verification oracle to accept a message/tag pair other than one generated by the tag generation oracle. We assume without essential loss of generality that the tag generation oracle chooses the nonces.

Theorem 2. *An adversary with forgery advantage A_f against GMAC has a distinguishing advantage A_{GPRF} against GPRF of at least $A_f - \lceil l/128 \rceil 2^{-t}$.*

An immediate corollary is that $A_{PRP} \geq A_f - \lceil l/128 \rceil 2^{-t} - q^2 l^2 2^{-142} + q^2 l^3 2^{-147}$.

5 The Finite Field $GF(2^{128})$

We define our field using a polynomial basis. We review this field representation in Appendix B to provide background on Galois fields and a guide to the implementer. In this representation, each element of $GF(2^{128})$ is represented by a polynomial with binary coefficients, and the 128 bits of the field element are identified with the 128 coefficients of the polynomial. These polynomials have the form $x_{127}\alpha^{127} + x_{126}\alpha^{126} + \dots + x_2\alpha^2 + x_1\alpha + x_0$, where each coefficient x_i is in $GF(2)$. We use x_i to denote both the i^{th} coefficient of the polynomial and the i^{th} bit of the field element. The variable α is a ‘dummy’ variable; we use it only to define our polynomials. Because we are interested in algebraic manipulations of polynomials, rather than the evaluation of polynomials, the value of α is irrelevant. We use the finite field defined by the field polynomial $\alpha^{128} + \alpha^7 + \alpha^2 + \alpha + 1$ which has low weight and terms of low order, properties that promote efficient implementations². The two field operations that we use are addition (which is identical with bitwise exclusive-or) and multiplication. The straightforward method for the multiplication in $GF(2^{128})$ is described in Appendix B. There are many other algorithms for multiplication in this field. We consider hardware and software implementations separately below. It is worth noting that, in the computation of $H(K, M)$, we repeatedly multiply an arbitrary field element M by another element K that remains fixed as long as the key is fixed.

Note that some implementations considered will not necessarily have a constant running time, due to conditional operations (usually conditional XORs that are performed if a particular bit is set). In such cases, the time variations constitute a timing channel that may leak information about the original message. Particularly, an attacker may be able to learn the number of bits set to one in a message simply by calculating the time taken to MAC the plaintext. This attack is only viable when the data being MACed needs to be secret to the attacker. In an encrypt-then-authenticate scheme where GMAC is the authentication component (e.g., in GCM mode), this is not a concern. However, if the MAC is implemented for stand-alone use instead of being implemented as a

² This polynomial was referenced from the *Table of Low-Weight Binary Irreducible Polynomials*, Gadiel Seroussi, Computer Systems Laboratory, HPL-98-135, August, 1998.

dedicated component of GCM or some similar mode, then implementors should take care to ensure that the MAC operation runs in constant time.

Also, as noted in [9], multiplication of a polynomial can be implemented efficiently in a parallel or pipelined manner by treating the polynomial as the sum of multiple polynomials, and then adding the results together once each individual polynomial has been computed. Particularly, one can calculate the sum of the terms where the key is raised to an even power separately from the sum of the terms where the key is raised to an odd power, add the two values together and get the correct result. This effectively amounts to processing the even numbered blocks of a message and the odd numbered blocks of a message independently.

5.1 Hardware

Binary field operations are especially suitable for hardware implementations. Many implementation strategies are discussed in the literature. Parr [11] summarizes the efficiency of various finite field multiplication methods for $GF(2^q)$ as follows:

Method	Time	Area
Parallel	1	$\mathcal{O}(q^2)$
Digit Serial [14]	q/D	$\mathcal{O}(qD)$
Bit Serial	q	$\mathcal{O}(q)$
Super Serial	qE	$\mathcal{O}(q/E)$

With $q = 128$, the parallel method is practical, and the digit serial method may provide a worthwhile tradeoff. Interestingly, the literature on Galois field implementations focuses on their use in elliptic curve cryptography, and there may be unexplored design alternatives that would provide advantages to their use in polynomial hashing. In an implementation of GCM, the polynomial hash would need to keep up with a pipelined implementation AES counter mode. This means that the number of clock cycles per $GF(2^{128})$ multiplication would need to be no larger than the number of cycles required to compute AES.

The use of a 128-bit field in the polynomial hash simplifies the implementation of GCM. Both the cipher and the MAC operate on data elements of the same width, eliminating the need to buffer data elements and match clocks. This is an appealing property, because the use of the larger field provides better security as well.

A typical pipelined implementation of a single AES encryption or decryption engine requires approximately 90K gates. In the same environment a straightforward implementation of GCM mode requires only an AES encryption unit plus approximately 30K gates for the hash function. When a single encryption unit is sufficient (e.g., half-duplex communication) OCB mode requires approximately 180K gates, needing both an encryption unit and a separate decryption unit. In contrast, CWC mode requires about 90K gates for encryption, along with 100K+ gates for implementing integer multiplication modulo 2^{127-1} .

5.2 Software

There are several software implementation strategies for multiplication in $GF(2^{128})$ that trade off between precomputed storage and efficiency. A naive implementation of the hash function would use no precomputation, recomputing the powers of the key for every block of input and testing each bit of the state individually. Such an implementation strategy is not overly efficient, running at speeds up to about 200 cycles, depending on the platform.

Of course, precomputation is often acceptable in software, depending on memory requirements. We examined several key-dependent strategies for precomputation. Shoup [13] notes that multiplying a value by a fixed element of $GF(2^{128})$ is a linear operation over $\{0, 1\}^{128}$. Therefore, we can group bits and precompute the effect that group of bits would have on the final result. For example, an element in $GF(2^{128})$ can be expressed as a vector of sixteen octets. We use the notation X_i to indicate the i^{th} octet of X . The multiplication $Y = X \cdot X$ of an element X times a fixed element K can be computed as

$$Y \leftarrow T0[X_0] \oplus T1[X_1] \oplus \dots \oplus T15[X_{15}] \quad (8)$$

Here the tables $T0, T1, T2, \dots, T15$ each map eight bits of input to one of 256 possible 16 octet intermediate outputs. The final output is determined by looking up each input byte in the appropriate table and then XORing all of the 16-octet outputs together. The total amount of storage required for these tables is $16 \times 16 \times 256 = 65,536$ bytes.

Using this technique, the $GF(2^{128})$ multiplication requires 4 fetches and 4 exors per byte on a 32-bit platform. The hash function requires an additional .25 exors per byte to perform the necessary addition. In contrast, AES requires 12 exors and 10 fetch operations per byte and the $GF(2^{127-1})$ multiply in CWC-HASH requires a minimum of 1.33 multiplies, .67 shifts and approximately 1.83 adds with carry per byte (there is slightly more work if no add with carry exists). The multiplies will generally be significantly more expensive than other operations (including fetches), and some machines may not have enough registers to store all the necessary intermediate values (particularly x86 machines). On 64-bit platforms, the $GF(2^{128})$ multiplication requires 2 fetches and 2 exors per byte, whereas the CWC-HASH multiply requires a minimum of .33 multiplies, .33 shifts and .33 adds with carry, with register allocation being less of an issue.

Due to the expensive latency involved with multiply operations on most platforms, we expect that GHASH using Shoup's method will be quicker than CWC-HASH on all platforms 32 bits or smaller. On 64-bit platforms, CWC-HASH should generally perform a bit better.

The problem with the above instance of Shoup's method is that the resulting table size is a bit large. The table size can be reduced to about 8K by keeping 32 tables mapping 4 bits of data to 16 possible outputs. This requires mapping 8 bit values to two four-bit values, which can either be done with shifting and masking or with lookup in a key independent 512-byte table.

One can implement GHASH with key-independent precomputation by mapping operations into a proper subfield, such as $GF(2^{16})$, as described by De Win et al. [6]. We did not experiment with this implementation technique.

We have compared the performance of C-based implementations of GHASH and CWC-HASH for various platforms. For GHASH, we tested with 64K key-dependent tables, 8K key-dependent tables and no tables. For CWC-HASH, we used implementations based on integer multiply. On 32-bit x86 platforms, where some speedup can potentially be seen using floating point multiplications, as discussed by Bernstein [5]. However, we were unable to achieve speedups using Gladman’s implementation of those techniques. All tests were compiled using the command line `gcc -O3 -funroll-loops -fexpensive-optimizations`. Results are shown in Table 1. In all cases, implementations of GHASH that use caching run in time comparable to CWC-HASH. GHASH with 64K caching performed marginally better than CWC-HASH on 32-bit platforms, whereas CWC-HASH performed better on 64-bit platforms. On platforms with smaller word sizes, GHASH will provide significant speed advantages over CWC-HASH.

In general, efficient implementations of these hash functions operate in fewer cycles than comparable AES implementations (though CWC-HASH becomes slower on platforms with words smaller than 32-bits). For example, all C-based AES implementations we have tested run above 20 cpb on common desktop platforms. As a result, dual use modes built on these hash functions perform acceptably well, particularly compared with standard approaches based on composition.

CPU	GHASH speeds			CWC-HASH
	64K tables	8K tables	No tables	
G4 1Ghz	13.1 cpb	17.3 cpb	119 cpb	13.7 cpb
G5 2Ghz (32-bit code)	9.60 cpb	13.3 cpb	99.6 cpb	15.3 cpb
G5 2Ghz (64-bit code)	8.21 cpb	9.56 cpb	123 cpb	6.02 cpb
Xeon 2.8 Ghz	14.8 cpb	27.6 cpb	205 cpb	31.4 cpb
Athlon XP 2500	12.8 cpb	24.5 cpb	140 cpb	21.3 cpb

Table 1. The results of timing tests on various platforms

6 Using GMAC as an incremental MAC

An *incremental* MAC is one that allows a new message/tag pair to be efficiently computed after part of a message is changed. This property is useful when the amount of data covered by the MAC is large and dynamic, and especially when it is remote. As a motivating example, consider the goal of providing strong message authentication on a data store. We want to periodically authenticate the data (e.g. at boot time), yet the data may be slowly changing as entries are changed or added. In many cases it may not be desirable to run the message

authentication code over the entire data store. The data may be extremely large, or perhaps stored at a remote network location, or we may not want to incur the latency cost from associating an expensive computation with each write operation. In these cases, an incremental MAC is advantageous.

GMAC can be used as an incremental MAC. Consider the case in which a message M has been protected using a nonce N , producing the tag T , then the message is changed to M' . We can produce a new tag T' corresponding to the new message M' and a new nonce N' as follows:

$$T' = T \oplus \text{len}(M') \oplus \text{len}(M) \oplus \bigoplus_{i \in \Delta} \delta M_i \cdot K^i \oplus F(K, N') \oplus F(K, N)$$

where $\delta M_i = M_i \oplus M'_i$ captures the changes between the two messages, and the set $\Delta = \{j : \delta M_j \neq 0\}$ contains all the indices corresponding to blocks that have changed. When $\#\Delta$ is considerably smaller than n , this ‘delta’ method offers a significant advantage.

In order to use the delta method, it is necessary to compute the powers of K corresponding to the indices in Δ . The value K^j can be computed by using the method of repeated squaring, which requires no more than 32 squares and multiplies in $GF(2^{128})$. An especially appealing property of $GF(2^{128})$ in this algorithm is the fact that the squaring operation in that field is simpler than the multiplication of two arbitrary elements. In practice, it is likely that many of the indices in Δ will form a continuous sequence, in which case only the initial power of K need be computed. A data format consisting of data blocks whose lengths are multiples of 16 octets would lend itself to use with the delta method.

The detailed design of a system that takes advantage of this method is outside the scope of this paper. Additional design or analysis work may need to be done before GMAC can be applied to some storage-security situations. However, the fact that it has the basic properties needed in these applications is a unique advantage of its design.

References

1. M. Bellare, A. Desai, E. Jorjipii and P. Rogaway, *A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation*, in Proceedings of 38th Annual Symposium on Foundations of Computer Science, IEEE, 1997.
2. M. Bellare, O. Goldreich, and S. Goldwasser, *Incremental cryptography: the case of hashing and signing*, CRYPTO 1994.
3. M. Bellare, R. Guerin, and P. Rogaway, *XOR MACs: New methods for message authentication using finite pseudorandom functions*, CRYPTO '95.
4. M. Bellare, P. Rogaway, and D. Wagner, *A conventional authenticated-encryption mode*, 2003. Available online at <http://eprint.iacr.org/2003/069/>.
5. D. Bernstein. *Floating-point arithmetic and message authentication*, 2000. Available online at <http://cr.yp.to/papers.html#hash127>.

6. De Win, Bosselaers, Vandenberghe, De Gersem, Vandewalle, *A Fast Software Implementation for Arithmetic Operations in $GF(2^n)$* , ESAT-COSIC Technical Report, 1997.
7. V. Gligor and P. Donescu, *Fast encryption and authentication: XCBC encryption and XECB authentication modes*, FSE 2001.
8. C. Jutla. *Encryption modes with almost free message integrity*. EUROCRYPT 2001.
9. T. Kohno, J. Viega, and D. Whiting, *The CWC-AES Dual-use Mode*, Internet Draft, Crypto Forum Research Group, May 20, 2003. Work in progress. Available online at <http://www.zork.org/cwc/draft-irtf-cfrg-cwc-01.txt>.
10. H. Krawczyk, *The Order of Encryption and Authentication for Protecting Communications*. In Lecture Notes in Computer Science, volume 2139, 2001.
11. C. Parr, *Implementation Options for Finite Field Arithmetic for Elliptic Curve Cryptosystems*, ECC '99.
12. P. Rogaway, M. Bellare, J. Black, and T. Krovitz, *OCB: a block-cipher mode of operation for efficient authenticated encryption*, ACM CCS 2001.
13. V. Shoup, *On Fast and Provably Secure Message Authentication Based on Universal Hashing*, CRYPTO '96.
14. L. Song, K.K. Parhi, *Efficient Finite Field Serial/Parallel Multiplication*, 1996 International Conference on Application-Specific Systems, Architectures, and Processors (ASAP '96), August 19 - 23, 1996.
15. M. Wegman and L. Carter. *New hash functions and their use in authentication and set equality*. Journal of Computer and System Sciences, 22:265279, 1981.
16. D. Whiting, N. Ferguson, and R. Housley. *Counter with CBC-MAC (CCM)*. Submission to NIST, 2002. Available online at <http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/>.

Appendix A: Security

The security of GMAC follows from that of GPRF, which follows from the security of the underlying block cipher. The single conjecture that we make, that the block cipher is indistinguishable from a PRP, is necessary in order to build a practical authentication or encryption algorithm. In GMAC, this single conjecture is sufficient as well. To paraphrase Occam, we do not multiply conjectures beyond necessity, and thus we minimize the probability that our security will be undermined by a conjecture that proves not to merit our trust.

In the following, we first prove some properties about the underlying hash functions, then prove the security of GPRF, then that of GMAC. We use the standard definitions of security of a MAC, a PRF, and a PRP, and we make use of the theory of universal hashing.

Universal Hashing

A function $g(K, M)$ is ϵ -almost xor universal if

$$\mathbf{P}[g(K, M) \oplus g(K, M') = a \mid K \stackrel{R}{\leftarrow} GF(2^{128})] \leq \epsilon \text{ for all } M \neq M' \text{ and } a. \quad (9)$$

Here the expression $\mathbf{P}[E \mid F]$ denotes the probability of the event E given that the event F has occurred, and the expression $K \stackrel{R}{\leftarrow} GF(2^{128})$ means that K is chosen at random from the set $GF(2^{128})$. We diverge slightly from the usual definition in order to allow our exposition to be more explicit³. We show that the hash underlying GPRF has the properties that we want by first showing that it is almost xor universal over most of its inputs.

Lemma 1. *The function H defined by Equation 4 is $\lceil l/128 + 1 \rceil 2^{-128}$ -almost xor universal when its inputs are restricted to lengths of l or fewer bits and values other than 96.*

Proof. Consider two distinct inputs M, M' . We assume without loss of generality that $\text{len}(M) \geq \text{len}(M')$. We consider the probability of the event that $H(K, M) \oplus H(K, M') = a$. When this event occurs, it follows that

$$\left(\bigoplus_{i=1}^{n'} (M_i \oplus M'_i) \cdot K^{n-i} \oplus \bigoplus_{i=n'+1}^n M_i \cdot K^{n-i} \right) \oplus (0^{64} \parallel (\text{len}(M) \oplus \text{len}(M'))) \cdot K = a. \quad (10)$$

³ In the standard definition, g would define a hash function family, and the selection of a key K would choose a particular hash function from that family of functions. Also, we assume a form for hash keys that is specific to our construction.

This condition can be re-expressed as $C(K) = 0$, where the polynomial C of degree n over $GF(2^{128})$ is defined by $C = \bigoplus_{i=1}^{n+1} C_i \cdot K^i$ with the coefficients

$$C_i = \begin{cases} a & \text{for } i = 0, \\ (0^{64} \parallel (\text{len}(M) \oplus \text{len}(M'))) & \text{for } i = 1, \\ M_i \oplus M'_i & \text{for } 2 < i \leq n' + 1, \\ M_i & \text{for } n' + 1 < i \leq n + 1. \end{cases} \quad (11)$$

Note that appending zero bits to message M changes the polynomial, not in the last term but in C_1 . There are at most $n+1$ values of $K \in G$ for which $C(K) = 0$ holds. This follows from the fact that an d^{th} degree polynomial over $GF(2^{128})$ has at most d distinct roots (this is the fundamental theorem of algebra over a finite field). The probability that $C(K) = 0$ holds, given that K is chosen at random from $GF(2^{128})$, is $(n+1)/2^{128} \leq \lceil l/128 + 1 \rceil 2^{-128}$. \square

We can now demonstrate the interesting properties of the function H .

Lemma 2. *The function H defined by Equation 4 obeys the bound*

$$\mathbf{P}[H(K, M) = \mathbf{incr}^j(H(K, M')) \mid K \xleftarrow{R} GF(2^{128})] \leq \lceil l/128 + 1 \rceil 2^{-128}, \quad (12)$$

where the inputs M and M' are distinct and are allowed to have any lengths up to l bits.

Proof. We call the event that $H(K, M) = \mathbf{incr}^j(H(K, M'))$ as E , for short. We separately consider the following cases:

1. $\text{len}(M) = \text{len}(M') = 96$,
2. $\text{len}(M) = 96$ and $\text{len}(M') \neq 96$, and
3. $\text{len}(M) \neq 96$ and $\text{len}(M') = 96$, and
4. $\text{len}(M) \neq 96$ and $\text{len}(M') \neq 96$.

In case 1, $H(K, M) = M \parallel 0^{32}$ and $H(K, M') = M' \parallel 0^{32}$. Because of the fact that M and M' are distinct and the fact that the increment function does not alter the leftmost 96 bits, the probability of a collision is zero. In case 2 when E occurs it follows that

$$\begin{aligned} H(K, M') &= \mathbf{incr}^{-j}(M \parallel 0^{32}) \\ &= A \parallel 0^{32} \text{ for some value } A \in \{0, 1\}^{96}. \end{aligned} \quad (13)$$

This condition can be re-expressed as $C(K) = 0$, where the polynomial C of degree n over $GF(2^{128})$ is defined by $C = \bigoplus_{i=1}^{n+1} C_i \cdot K^i$ with the coefficients

$$C_i = \begin{cases} A \parallel 0^{32} & \text{for } i = 0, \\ (0^{64} \parallel \text{len}(M')) & \text{for } i = 1, \\ M'_i & \text{for } 2 < i \leq n' + 1, \end{cases} \quad (14)$$

There are at most n values of $K \in G$ for which $C(K) = 0$ holds, since that it is the number of distinct roots of C . This fact is true for any value A . Thus the probability of E , given that K is chosen at random from $GF(2^{128})$, is $(n' + 1)/2^{128} \leq \lceil l/128 + 1 \rceil 2^{-128}$. Case 3 is identical to case 2 after a relabeling of inputs. Case 4 is identical to Lemma 1. The claimed result holds in all four cases. \square

Lemma 1 shows that the initial counter values C_0 in GRPF, corresponding to the different nonces, have a high probability of being distinct because the set of nonces provided as inputs to that algorithm are distinct. This leads directly to a proof of security of that function.

GPRF

The proof of Theorem 1 establishes the security of GPRF based on the indistinguishability of the PRP E from a random permutation.

Proof of Theorem 1. For each nonce provided to GPRF, there is a sequence of counter values used as the input to E . We define the event U as the case in which $b = 1$ and all of these counter values are distinct, across all counter sequences. The advantage can be expressed as

$$\begin{aligned} A_{\text{GPRF}} &= \mathbf{P}[d = 1 \mid b = 1, U] \mathbf{P}[U] + \mathbf{P}[d = 1 \mid b = 1, \neg U] \mathbf{P}[\neg U] - \mathbf{P}[d = 1 \mid b = 0] \\ &= (\mathbf{P}[d = 1 \mid b = 1, U] \mathbf{P}[U] - \mathbf{P}[d = 1 \mid b = 0]) + \mathbf{P}[d = 1 \mid b = 1, \neg U] \mathbf{P}[\neg U] \\ &\leq (\mathbf{P}[d = 1 \mid b = 1, U] - \mathbf{P}[d = 1 \mid b = 0]) \mathbf{P}[U] + \mathbf{P}[d = 1 \mid b = 1, \neg U] \mathbf{P}[\neg U] \\ &\leq (\mathbf{P}[d = 1 \mid b = 1, U] - \mathbf{P}[d = 1 \mid b = 0]) + \mathbf{P}[d = 1 \mid b = 1, \neg U] \mathbf{P}[\neg U]. \end{aligned}$$

Note that the term $\mathbf{P}[d = 1 \mid b = 1, U] - \mathbf{P}[d = 1 \mid b = 0]$ is equal to the advantage A_{PRF} of distinguishing the counter mode of a E from a truly random function, in the case that all counter values are distinct. The value $\mathbf{P}[d = 1 \mid b = 1, \neg U]$ in the last term is no greater than one. Below we assume that it is equal to one, and rely on the fact that $\mathbf{P}[\neg U]$ is extremely small to keep the advantage low.

We next consider the probability of the event U . For each nonce provided to GPRF, there is a sequence of counters used as the input to E . For a nonce N , the sequence is equal to

$$H(K, N), \mathbf{incr}(H(K, N)), \mathbf{incr}^2(H(K, N)), \dots, \mathbf{incr}^{l-1}(H(K, N)), \quad (15)$$

where we use the notation that $\mathbf{incr}^j()$ indicates j repeated applications of the increment function. The event U occurs when all counter sequences are distinct. When the counter sequences for nonce N and nonce N' overlap, then

$$H(K, N) = \mathbf{incr}^j(H(K, N')) \text{ for some } j \text{ such that } -l' < j < l'. \quad (16)$$

Here we use the shorthand that $l' = \lceil l/128 \rceil$ indicates the number of counters in each sequence. From Lemma 2, we know that the probability of this event,

for each of the $2^{l'} - 1$ values of j , is equal to $\lceil l/128 + 1 \rceil 2^{-128}$. Thus $\mathbf{P}[U] \leq 1 - \lceil l/128 + 1 \rceil (2q \lceil l/128 \rceil)^2 2^{-128}$, and

$$A_{\text{GPRF}} \leq A_{\text{PRF}} + q^2 \lceil l/128 + 1 \rceil^3 2^{-126},$$

where we make use of the identities noted above.

From [1], we know that the advantage A_{CTR} of an adversary against counter mode of a PRP is bounded by $A_{\text{CTR}} \leq A_{\text{PRP}} + c^2 2^{-128}$, where A_{PRP} is the adversary's advantage in distinguishing the PRP from a random permutation, and c is the number of blocks of output that the adversary can see. In our case, the number of blocks is $c = q \lceil l/128 \rceil$. Thus

$$A_{\text{GPRF}} \leq A_{\text{PRP}} + q^2 (\lceil l/128 \rceil^2 2^{-128} + \lceil l/128 + 1 \rceil^3 2^{-126}), \quad (17)$$

and the result claimed follows. \square

The advantage against GPRF is roughly like that against the standard counter mode, with a quadratic term that becomes important when the number of bits output is about 2^{71} . However, it also has a term that is quadratic in q but cubic in l , which is due to the fact that collisions in the underlying hash are more likely when the lengths of the messages that are hashed becomes greater. This term becomes important whenever $2^{71}/l$ is close to one, and is thus the dominant term. The implication on the user of GPRF is that when long outputs are generated, keys must be changed more often. Note that if, for some particular key, GPRF is used only with 96-bit nonces, then the second term in Equation 7 vanishes, and the security of GPRF is exactly that of counter mode as in [1].

GMAC

To analyze the security of GMAC, we first establish the important properties of GHASH.

Lemma 3. *The underlying hash function H is $\lceil l/128 \rceil 2^{-t}$ -almost xor universal when its inputs are restricted to lengths of l or fewer bits.*

Proof. This proof follows that of Lemma 1. Consider two distinct inputs $(A, C), (B, D)$. The hash algorithm operates on the decompositions of these bit strings

$$\begin{aligned} A \parallel 0^{l_A} &= A_1 \parallel A_2 \parallel \cdots \parallel A_{n_A} \\ C \parallel 0^{l_C} &= C_1 \parallel C_2 \parallel \cdots \parallel C_{n_C} \\ B \parallel 0^{l_B} &= B_1 \parallel B_2 \parallel \cdots \parallel B_{n_B} \\ D \parallel 0^{l_D} &= D_1 \parallel D_2 \parallel \cdots \parallel D_{n_D} \end{aligned}$$

We consider the probability of the event that the hash of (A, C) , XORed with the hash of (B, D) , is equal to the t -bit value a . When this event occurs, it follows

that $C(K) = 0$, where the polynomial C of degree $n = \max(n_A + n_C, n_B + n_D)$ over $GF(2^{128})$ is defined by $C = \bigoplus_{i=0, n+1} C_i \cdot K^i$ with the coefficients

$$C_i = \begin{cases} a' & \text{for } i = 0, \\ \text{len}(A) \oplus \text{len}(B) \parallel \text{len}(C) \oplus \text{len}(D) & \text{for } i = 1, \\ X_i & \text{for } 2 \leq n + 1, \end{cases} \quad (18)$$

where X_i represents a linear combination of the blocks of A, B, C and D in which each block appears exactly once. Here a' is any element of the set A that contains all of the elements of $GF(2^{128})$ that have their rightmost $128 - t$ bits equal to zero.

For any fixed value of a' , there are at most n values of $K \in G$ for which $C(K) = 0$ holds. This follows from the fact that an n^{th} degree polynomial over $GF(2^{128})$ has at most n distinct roots. The probability that $C(K) = 0$ holds, given that K is chosen at random from $GF(2^{128})$, is $n/2^{128}$. The collision event occurs whenever $C(K) = 0$, for *any* value of $a' \in A$. There are 2^{128-t} elements in the set A , so $\mathbf{P}[H(K, M) \oplus H(K, M') \mid K \xleftarrow{R} GF(2^{128})] \leq n2^{-t}$. This fact is true for all n and n' such that $n' \leq n \leq \lceil b/128 \rceil$, and lemma 3 follows. \square

Proof of Theorem 2. Assume that we have a machine that generates message/tag pairs with forgery advantage A_f , using a chosen-message attack. We use this machine to distinguish between GPRF ($b = 1$) and a random function ($b = 0$) as follows. We use the oracle to the black box function defined by the selection of b to implement GMAC, then run the machine and provide it with access to a tag-generation oracle and a message/tag verification oracle. If the machine succeeds in forging a message/tag pair, then $d = 1$; otherwise, $d = 0$. When $b = 0$, the best strategy for forging messages is to choose values of M, M' and δT such that $\mathbf{P}[\text{GHASH}(K, M') \oplus \text{GHASH}(K, M) = \delta T \mid K \xleftarrow{R} GF(2^{128})]$ meets the lower bound in Lemma 3. The probability of success in this case is no greater than $\lceil l/128 \rceil 2^{-t}$, so $\mathbf{P}[d = 1 \mid b = 0]$ equals that value. When $b = 1$, the probability that $d = 1$ is just A_f . Thus the distinguishing advantage A_{GPRF} is bounded by $A_{\text{GPRF}} \geq A_f - \lceil l/128 \rceil 2^{-t}$. \square

Appendix B: Multiplication in $GF(2^{128})$

A finite field is defined by its multiplication and addition operations. These operations obey the basic algebraic properties that one expects from multiplication and addition (commutativity, associativity, and distributivity). Both operations map a pair of field elements onto another field element. In a polynomial basis, the multiplication of two elements X and Y consists of multiplying the polynomial representing X with the polynomial representing Y , then dividing the resulting 256-bit polynomial by the field polynomial; the 128-bit remainder is the result. We describe multiplication in more detail below. The addition of two elements X and Y consists of adding the polynomials together. Because each coefficient

is added independently, and the coefficients are in $GF(2)$, this operation is identical to the bitwise exclusive-or of X and Y . No reduction operation is needed. Subtraction over $GF(2^{128})$ is identical to addition, because the field $GF(2)$ has that property.

To describe multiplication, we take the small first step of showing how to multiply a field element X by the field element P defined by

$$P_i = \begin{cases} 1 & \text{for } i = 1 \\ 0 & \text{otherwise,} \end{cases} \quad (19)$$

then we show how to use this method to multiply any two field elements. The element P corresponds to the polynomial α . The multiplication of a polynomial by α is simple; it corresponds to a shift of indices:

$$\alpha \cdot (x_{127}\alpha^{127} + x_{126}\alpha^{126} + \dots + x_1\alpha + x_0) = x_{127}\alpha^{128} + x_{126}\alpha^{127} + \dots + x_1\alpha^2 + x_0\alpha. \quad (20)$$

Now we must divide the result by the field polynomial and find the remainder. We denote the field polynomial as f . To find the remainder of a polynomial $\alpha^{128} + a$, where $a = a_{127}\alpha^{127} + a_{126}\alpha^{126} + \dots$, we need to find polynomials q and r such that $\alpha^{128} + a = q \cdot f + r$, where q is the quotient and r is the remainder. We can solve this equation for r when $q = 1$:

$$r = \alpha^{128} + a - f = a + \alpha^7 + \alpha^2 + \alpha + 1. \quad (21)$$

The highest term of f is canceled away, and the net effect is just to add the lowest terms of f to a . To compute $Y = X \cdot P$, we combine the two steps of shifting coefficients and adding in the lowest terms of f if the highest term of X is equal to one. In bit operations, this can be expressed as

```

if  $X_{127} = 0$  then
   $Y \leftarrow \text{leftshift}(X)$ 
else
   $Y \leftarrow \text{leftshift}(X) \oplus Z$ 
end if

```

where Z is the element whose rightmost eight bits are 10000111, and whose leftmost 120 bits are all zeros.

In order to multiply two arbitrary field elements X and Y , we can express Y in terms of P , then use the algorithm described above. This is expressed in the following algorithm, which takes X and Y as inputs and returns their product.

```

 $Z \leftarrow 0, V \leftarrow X$ 
for  $i = 0$  to 127 do
  if  $Y_i = 1$  then
     $Z \leftarrow Z \oplus V$ 
  end if
   $V \leftarrow V \cdot P$ 

```

end for
return Z

In this algorithm, V runs through the values of $V, V \cdot P, V \cdot P^2, \dots$, and the powers of P correspond to the powers of α .