AES-Based Authenticated Encryption Modes in Parallel High-Performance Software

Andrey Bogdanov and Martin M. Lauridsen and Elmar Tischhauser

Department of Mathematics, Technical University of Denmark, Denmark
{anbog,mmeh,ewti}@dtu.dk

Abstract. Authenticated encryption (AE) has recently gained renewed interest due to the ongoing CAESAR competition. This paper deals with the performance of block cipher modes of operation for AE in parallel software. We consider the example of the AES on Intel's new Haswell microarchitecture that has improved intructions for AES rounds and finite field multiplication.

As opposed to most previous high-performance software implementations of operation modes – that have considered the encryption of single messages – we propose to process multiple messages in parallel. We demonstrate that this message scheduling is of significant advantage for most modes. As a baseline for longer messages, the performance of AES-CBC encryption on a single core increases by factor 6.8 when adopting this approach.

For the first time, we report optimized AES-NI implementations of the novel AE modes OTR, McOE-G, COBRA, and POET – both with single and multiple messages. For almost all AE modes considered, we obtain a consistent speed-up when processing multiple messages in parallel. Notably, among the nonce-based modes, AES-CCM gets by factor 3.5 faster and its performance is about 1.2 cpb which is close to that of AES-GCM (the latter, however, possessing classes of weak keys), with AES-OCB3 still performing at only 0.69 cpb. Among the nonce-misuse resistant modes, AES-McOE-G receives a speed-up by factor 4 and its performance is about 1.44 cpb, which is faster than AES-COBRA with its 1.55 cpb but slower than AES-COPA with 1.29 cpb.

Keywords. authenticated encryption, AES-NI, `pclmulqdq`, COBRA, COPA, GCM, McOE-G, OCB3, OTR, POET

1 Introduction

An *authenticated encryption (AE) scheme* is a secret-key cryptographic primitive which combines encryption and authentication. Since around 2000, interest in such schemes has been on the rise. Quite some AE schemes have been proposed, and more are expected to join the ranks with the ongoing CAESAR[1] cryptographic competition [1]. The CAESAR competition is a move towards selecting a portfolio of AE schemes that should improve upon the state of the art.

Arguably the most common way of constructing an AE scheme is a mode of operation for a block cipher [2,6,7,9,10,13,26,27,29,32–35]. Other approaches are also feasible, e.g. the hash function based Hash-CFB [15] or the permutation based SpongeWrap [8] and APE [5]. When realizing an AE mode of operation in practice using a block cipher, the tried-and-true choice of the underlying primitive naturally falls upon the AES. Indeed, GCM instantiated with the AES (denoted AES-GCM) is one such example which is widely used in e.g. TLS 1.2 [11] and included into e.g. the NSA Suite B Cryptography [3].

As authenticated encryption usually deals with processing bulk data, for any AE scheme, especially a standardized one, performance is of paramount importance. With AES being deployed in numerous cryptographic schemes and protocols on the Internet, among those in AES-GCM, Intel launched in 2011 their Westmere microarchitecture which, for the first time, implemented dedicated instructions for AES encryption and decryption as well as for multiplication in the finite field needed by GCM. Since, also AMD has introduced such

[1] Competition for Authenticated Encryption: Security, Applicability, and Robustness

instructions in its processor series. The newest Haswell microarchitecture by Intel introduced in 2013 has improved upon the efficiency of these intructions.

This paper deals with the performance of AES modes of operation for authenticated encryption on Intel Haswell.

Our Contributions. Our contributions are as follows:

- **Parallel processing of multiple messages:** The comminication devices of high-speed links are likely to process many messages at the same time: Indeed, on the Internet, the bulk of data is transmitted in packets of sizes betweeen 1 and 2 KB. While most previous implementations of modes of operation for block ciphers consider processing a single message, we propose to process several messages in parallel, which reflects this reality. For instance, the simultaneous processing of multiple messages is beneficial when using an inherently sequential mode. Additionally, also for many parallel modes, this brings clear advantages. While this type of message processing was briefly discussed in the paper proposing ALE [10], the consideration was limited to a single dedicated AE scheme based on AES components. In the paper at hand, for the first time, we deal with a variety of AE modes of operation in this setting. See Section 3.
- **Speeding up AES-CBC encryption by factor 6.5:** Whereas CBC decryption is essentially parallel, CBC encryption is serial, thus, not being able to profit from the available parallelism if a single message is processed only. However, when several messages are processed simultaneously, this parallelism can be exploited by the implementation. We report an AES-CBC implementation with AES-NI on Intel Haswell featuring 0.56 cycles per byte (cpb) for longer messages of 2048 bytes. This constitutes a speed-up by factor 6.8, which is an interesting show-case for the parallel processing of multiple messages. See Section 3.3 for a discussion of CBC encryption in this setting.
- **First AES-NI implementations of OTR, McOE-G, COBRA, and POET:** We provide AES-NI implementations both of recent nonce-based modes such as OTR and of novel nonce-misuse resistant modes such as McOE-G, COBRA and POET. We also present the first Haswell performance figures for AES-CCM, AES-OCB3, and AES-COPA that have been implemented with AES-NI only on older platforms such as Sandy Bridge and Ivy Bridge before. After the discussion of Haswell's AES-NI and binary field multiplication in Section 4, we present our new implementation results in Section 5.
- **AE modes of choice:** We provide a comprehensive performance study of both nonce-based and nonce-misuse resistant AE modes of operation both in the single and multiple message setting. This study establishes that when processing a long single message, in the nonce-based setting, OCB3 with its 0.63 cpb is the mode of choice except for very short messages, for which OTR performs better. Among the nonce-misuse resistant modes, COPA outperforms the other modes for all message lengths. When considering multiple messages of 2048 bytes, among the nonce-based modes, AES-CCM gets by factor 3.5 faster, at about 1.2 cpb being close to AES-GCM (the latter, however, possessing classes of weak keys), with AES-OCB3 still performing best at 0.69 cpb. Among the nonce-misuse resistant modes, AES-McOE-G receives a speed-up of factor 4 to about 1.44 cpb, with AES-COPA again performing best at 1.29 cpb. See Section 5.3 for a discussion.

2 Authenticated Encryption Modes

We split up our consideration into *nonce-based* AE modes, i.e. modes that require a unique input for each message to guarantee security, and *nonce-misuse resistant* AE modes, which guarantee a certain level of security even when repeating nonces. The modes we consider in the former camp are CCM, GCM, OCB3 and OTR, while the nonce-misuse resistant modes considered are McOE-G, COPA, COBRA and POET. Table 1 gives a comparison of the eight AE modes considered in this work.

The price to pay for a mode to be nonce-misuse resistant includes extra computation, a higher serialization degree, or both. One of the fundamental questions we answer in this work, is how much one has to pay in terms of performance, to maintain security when repeating nonces.

Table 1: Overview of the AE modes considered in this paper. The top half are nonce-based modes while the lower half are nonce-misuse resistant modes. The "Par" column indicates whether a mode is parallelizable. The "E^{-1}-free" column indicates whether a mode needs to call the inverse of the underlying block cipher in decryption/verification. The "E" and "M" columns give the number of calls per message block to the underlying n-bit block cipher and multiplications in $GF(2^n)$, respectively.

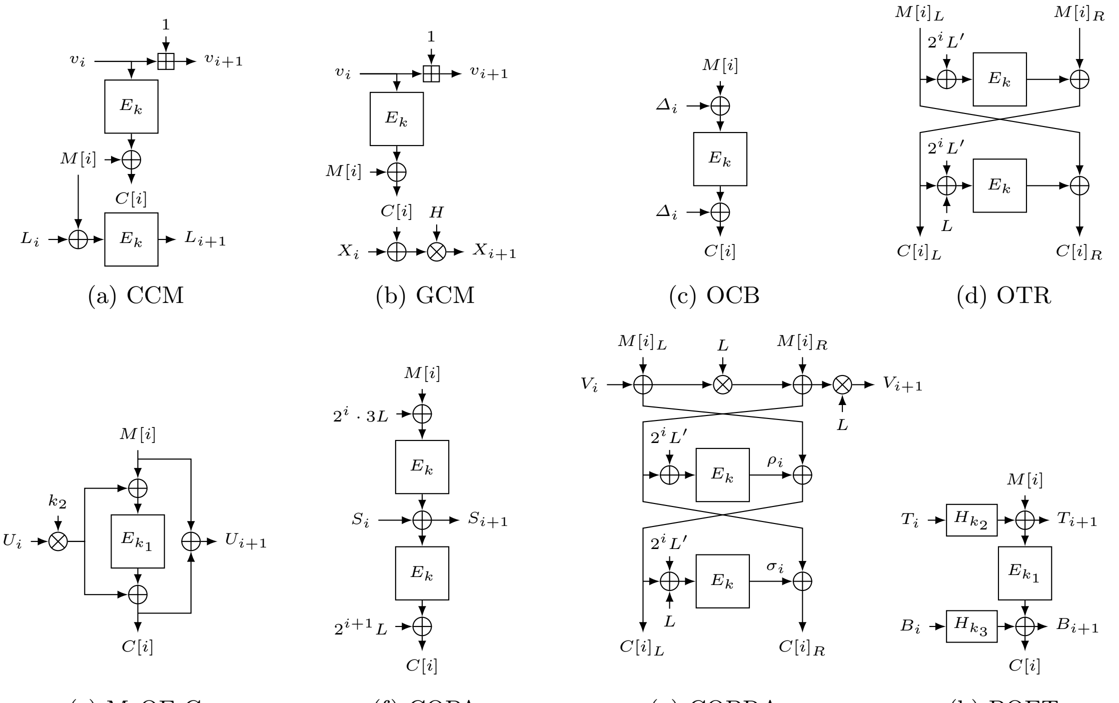
	Year	Par	E^{-1}-free	E	M	Description
Nonce-based AE modes						
CCM [34]	2002	–	yes	2	–	CTR encryption, CBC-MAC authentication
GCM [27]	2004	yes	yes	1	1	CTR mode with chain of multiplications
OCB3 [26]	2010	yes	–	1	–	Xor-encrypt-Xor (XEX) construction with doubling
OTR [29]	2013	yes	yes	1	–	Two-block Feistel structure
Nonce-misuse resistant AE modes						
McOE-G [12]	2011	–	–	1	1	Serial multiplication-encryption chain
COPA [6]	2013	yes	–	2	–	Two-round XEX
COBRA [7]	2014	yes	yes	1	1	OTR with chain of multiplications on top
POET [2]	2014	yes	–	1	2	XEX with two AXU (multiplication) chains

2.1 Specifications

Here, we briefly give specifications of the eight AE modes of Table 1. We use the notation that M and C are arrays of message and ciphertext, respectively. We use $M[i]$ and $C[i]$ to denote the ith block of these arrays with $1 \leq i \leq m$. Exceptions are COBRA and OTR where we define a block as twice the block size of the underlying block cipher, and we let $(M[i]_L \parallel M[i]_R)$ and $(C[i]_L \parallel C[i]_R)$ denote the ith block of message and ciphertext, respectively. We use E_k to denote the encryption with the underlying block cipher using master key k and let N denote the nonce. Multiplications (denoted $\otimes$) and additions (denoted $\oplus$) are done in what we call the *GCM finite field* $GF(2^{128}) = GF(2)[x]/x^{128} + x^7 + x^2 + x + 1$. Where applicable, we let $\mathcal{A}$ denote the output of the processing of associated data.

CCM. The CCM mode (Counter with CBC-MAC, see Figure 1a) is a design by Ferguson et al. [34]. It is a part of IEEE 802.11i (as a variant), IPsec [21] and TLS 1.2 [11]. Initially, $L_1 = \mathcal{A}$ and v_1 depends on the nonce N (v_i is a counter incremented once per block). The tag is computed as $T = E_k(N) \oplus L_{m+1}$.

GCM. GCM (Galois/Counter Mode, see Figure 1b) is a design by McGrew and Viega [27]. At the time of writing, it is the most widely used authenticated encryption scheme, and is included in a range of specifications, including TLS 1.2 [11] and NSA Suite B Cryptography [3]. Initially, $v_1 = 1$ (v_i is a counter incremented once per block) and $X_1 = \mathcal{A} \cdot H$, where $H = E_k(0)$. The tag is computed as $T = E_k(0) \oplus H(X_{m+1} \oplus len(A) \| m)$ where $len(A)$ is the length of associated data.


(a) CCM (b) GCM (c) OCB (d) OTR

(e) McOE-G (f) COPA (g) COBRA (h) POET

Fig. 1: Processing of a message block (or two message blocks in the case of COBRA and OTR) for the AE modes benchmarked in this work

OCB3. OCB3 (see Figure 1c) is a mode by Krovetz and Rogaway [26]. It uses a table-based approach of pre-computing values $L_j = 2L_{j-1}, j \geq 1$, where $L_0 = 4E_k(0)$. Initially, $\varDelta_1 = \mathit{Init}(N) \oplus L_0$ where $\mathit{Init}(N)$ is a nonce-derived value, and $\varDelta_i = L_{ntz(i)} \oplus \varDelta_{i-1}$ for $i > 1$. The $ntz(i)$ function returns the number of trailing zeroes in the binary representation of i. The tag is computed as $T = E_k(\varSigma \oplus \varDelta_{m+1}) \oplus \mathcal{A}$, where $\varSigma = \bigoplus_{i=1}^{m} M[i]$.

OTR. The OTR mode is a design by Minematsu [29]. It uses a Feistel scheme for two consecutive blocks, but in OTR their processing is completely independent of two other consecutive blocks (see Figure 1d), allowing for good parallelization. We assume the number of blocks is a multiple of two. Under this assumption, the last two blocks of ciphertext are swapped (not shown in figure). Values used are $L = E_k(N)$ and $L' = 4L$. The tag os computed as $T = \mathcal{A} \oplus E_k(\varSigma \oplus 3 \cdot 2^{m/2-1} L')$, where $\varSigma = \bigoplus_{i=2,i \text{ even}}^{m} M[i] \oplus Z$ and Z is the output of the top encryption in the Feistel branch of the last two blocks.

McOE-G. The McOE-G mode [13] (see Figure 1e) is a member of the McOE family [12] by Fleischmann et al. from FSE 2012. The mode uses two keys, k_1 and k_2; one for encryption and one for multiplication. Initially, $U_1 = \tau \oplus N$, where $\tau = E_k(N \oplus \mathcal{A}) \oplus \mathcal{A}$. The tag is generated as $T = E_{k_1}(\tau \oplus (U_{m+1} \otimes k_2)) \oplus (U_{m+1} \otimes k_2)$.

COPA. COPA (see Figure 1f) is a design by Andreeva et al. from Asiacrypt 2013 [6]. Values used are $S_0 = \mathcal{A} \oplus L$ and $L = E_k(0)$. The tag is generated as $T = E_k(E_k(\varSigma \oplus 2^{m-1}3^2 L) \oplus S_{m+1})$, where $\varSigma = \bigoplus_{i=1}^{m} M[i]$.

COBRA. COBRA is a mode proposed by Andreeva et al. at FSE 2014 [7]. Similarly to OTR, it operates on two consecutive blocks in a Feistel scheme after the mixing with a chaining value V (see Figure 1g). For simplicity, COBRA is defined for an even number of blocks. Values used are $V_0 = L^2 \oplus N \cdot L$, $L = E_k(1)$ and $L' = 4L$. The tag is computed as $T = E_k(E_k(\varSigma \oplus 3(2^2 L' \oplus L)) \oplus N \oplus \mathcal{A} \oplus 3^2(2^2 L' \oplus L))$, where $\varSigma = \bigoplus_{i=1}^{m} \sigma_i \oplus \rho_i$.

POET. This mode (see Figure 1h) is a design by Abed et al. presented at FSE 2014 [2]. It keeps two running chaining values which we denote T and B, and uses these with the XEX paradigm by Rogaway. The mode uses three keys. The top and bottom hashing H_{k_j}, $j \in \{2,3\}$, and can be either multiplication by the key or encryption for four or ten rounds in the case of AES-128. For the purpose of our benchmarking, we consider for POET only the hash being multiplication in $GF(2^{128})$. Initially, $T_0 = B_0 = 1$. The tag is computed as $T = E_{k_1}(\tau \oplus T_{m+1}) \oplus B_{m+1}$ where $\tau = E_{k_1}(\mathcal{A}_T \oplus N) \oplus \mathcal{A}_B$. Here $\mathcal{A}_T$ and $\mathcal{A}_B$ the top and bottom parts of $\mathcal{A}$. We do not consider the other two variants of POET in this work: While using the 4-round AES instead of the multiplication is not a mode of operation, using the entire AES for hashing results in 3 block cipher calls per block of bulk data, which is considered rather inefficient.

3 Simultaneous Processing of Multiple Messages

3.1 Motivation

A substantial number of block cipher modes of operation for (authenticated) encryption are inherently serial in nature. This includes the classic CBC and CCM modes. Also, more recent designs essentially owe their sequential nature to design goals, e.g allowing lightweight implementations or achieving stricter notions of security, for instance not requiring a nonce for security (or allowing its reuse). Examples include ALE [10], APE [5], CLOC [23] the McOE family of algorithms [12, 13], and some variants of POET [2].

While being able to perform well in other environments, such algorithms cannot benefit from the available pipelining opportunities on contemporary general-purpose CPUs. For instance, as detailed in Section 4, the AES-NI encryption instructions on Intel's recent Haswell architecture feature a high throughput of 1, but a relatively high latency of 7 cycles. Modes of operation that need to process data serially will invariably be penalized in such environments.

Furthermore, even if designed with parallelizability in mind, (authenticated) modes of operation for block ciphers typically achieve their best performance when operating on somewhat longer messages, often due to the simple fact that these diminish the impact of potentially costly initialization phases and tag generation. Equally importantly, only longer messages allow high-performance software implementations to make full use of the available pipelining opportunities [4, 17, 26, 28].

In practice, however, one rarely encounters messages which allow to achieve the maximum performance of an algorithm. Recent studies on packet sizes on the Internet demonstrate that they basically follow a bimodal distribution [25, 30, 31]: 44% of packets are between 40 and 100 bytes long, and 37% are between 1400 and 1500 bytes in size. First, this emphasizes the importance of good performance for messages up to around 2 KB, as opposed to longer messages. Second, when looking at the weighted distribution, this implies that the vast majority of data is actually transmitted in packets of medium size between 1 and 2 KB.

Considering the first mode of the distribution, we observe that many of the very small packets of Internet traffic comprise TCP ACKs (which are typically not encrypted), and that the use of authentication and encryption layers such as TLS or IPsec incurs overhead significant enough to blow up a payload of 1 byte to a 124 byte packet [22].

It is therefore this range of message sizes (128 to 2048 bytes) that authenticated modes of encryption should excel at processing.

3.2 Multiple Message Processing

It follows from the above discussion that the standard approach of considering one message at a time, while arguably optimizing message processing latency, cannot always generate optimal throughput in high-performance software implementations in most practically relevant scenarios. This is not surprising for the inherently sequential modes, but even when employing a parallelizable design, the prevailing distribution of message lengths makes it hard to achieve the best performance.

In order to remedy this, we propose to consider the scheduling of multiple messages in parallel *already in the implementation of the algorithm itself*, as opposed to considering it as a (single-message) black box to the message scheduler.

This opens up possibilities of increasing the performance in the cases of both sequential modes and the availability of multiple shorter or medium-size messages. In the first case, the performance penalty of serial execution can potentially be hidden by filling the pipeline with a sufficient number of operations on independent data. In the second case, there is a potential of increasing performance by keeping the pipeline filled also for the overhead operations such as block cipher or multiplication calls during initialization or tag generation. Note that while in this paper we consider the processing of multiple messages on a single core, the multiple message approach naturally extends to multi-core settings.

Conceptually, the transition of a single message to a multiple message implementation can be viewed as similar to the transition from a straightforward to a bit-sliced implementation approach.

We also note that typically, many messages belonging to the same communication session will be encrypted under the same key (but with different nonces in case of a nonce-based mode). This idea has been introduced in the performance study of the dedicated AE algorithm ALE [10], without however being explored in its full generality.

Note that while multiple message processing has the potential to increase the throughput of an implementation, it can also increase its latency (see also next subsection). The amount of parallelism therefore has to be chosen carefully and with the required application profile in mind.

3.3 Speeding Up AES-CBC Encryption

The CBC mode of operation is one of the most widely employed ones, being used in virtually any protocol suite (SSH, TLS, IPsec, . . .). While CBC encryption is strictly sequential, CBC decryption *can* be parallelized [4].

In here, we demonstrate that significant speed-ups are possible for CBC encryption when processing multiple messages in parallel. Our sample target platform is Intel's Haswell architecture with AES-NI instructions, though the approach also works for other recent microarchitectures including Intel's Westmere, Sandy Bridge, and Ivy Bridge. The implementation is as follows: each step of the CBC encryption algorithm is executed for every message before continuing with the next step, thus allowing pipelining of the otherwise serial operations. We summarize the performance results for various choices for the number of multiple messages in Table 2. Along the performance data, we list the relative speed-up compared to a single-message implementation for each level of parallelism.

We observe that speed-ups of up to $\times 6.79$ are possible using 8 multiple messages. Since on Haswell, the theoretical limit is 7 by the latency of `aesenc` (see Section 4.1), this speed-up of CBC encryption can be considered almost optimal. Note that the speed-ups reported in [4] require the use of very long messages (32 KB), while the speed-ups of Table 2 are achieved at a realistic message size of 2 KB. Experiments show that similar speed-ups can be achieved in the multiple message setting for shorter messages as well.

Table 2: Performance of CBC encryption (cpb) and relative speed-up when processing multiple messages (2048 bytes).

		multiple messages						
	single msg.	2	3	4	5	6	7	8
AES-CBC	3.8	1.91	1.27	0.96	0.78	0.66	0.57	0.56
rel. speed-up	$\times 1.00$	$\times 1.99$	$\times 3.00$	$\times 3.96$	$\times 4.78$	$\times 5.76$	$\times 6.67$	$\times 6.79$

A point worth discussing is the loss in latency we have to pay for the multiple message processing. Since the speed-up is limited by the parallelization level, we can at most hope for the same latency as in the single-message case. Table 2 shows that this is actually achieved for 2 and 3 messages: Setting $|M| = 2048$, instead of waiting $3.8 \cdot |M|$ cycles in the single-message case, we have a latency of $1.91 \cdot 2 = 3.82|M|$ and $1.27 \cdot 3 = 3.81|M|$ cycles, respectively. Starting from 4 messages, the latency slightly starts to increase together with the throughput, however remaining at a manageable level even for 8 messages, where it is only around 18% higher than in the single-message case. This has to be contrasted (and, depending on the application, weighed against) the significant 6.8 times speed-up in throughput.

Finally, we note that this technique also allows accelerating the computation of CBC-MAC by the same factors and therefore serves as a basis to speed up CCM in the multiple message setting.

4 Intel's Haswell Microarchitecture

In this section, we describe the special Intel instructions that allow for fast AES implementations and multiplications in $GF(2^{128})$. We also provide an overview of the changes made in Haswell, compared to previous microarchitectures.

4.1 Improvements for the AES

AES is the block cipher of choice in modern systems, and is supported by virtually any new protocol using block ciphers. With such a widely used cipher, going to the limits performance-wise, both in hardware and software, is of paramount importance. With this realization, Intel proposed and implemented since their 2010 Westmere microarchitecture, special instructions for fast AES encryption and decryption [16]. The *AES New Instruction Set*, or *AES-NI* for short, comprises six CPU instructions: `aesenc` (one round of AES), `aesenclast` (last round of AES), their decryption equivalents `aesdec` and `aesdeclast`, `aesimc` (inverse `MixColumns`) and `aeskeygenassist` for a faster key scheduling. The instructions do not only offer better performance, but security as well, leaking no timing information.

AES-NI is supported in a subset of Westmere, Sandy Bridge, Ivy Bridge and Haswell microarchitectures. A range of AMD processors also support the instructions under the name *AES Instructions*, including processors in the Bulldozer, Piledriver and Jaguar series [20]. In Haswell, the AES-NI encryption and decryption instructions had their latency improved from 8 cycles on Sandy and Ivy Bridge, down to 7 cycles [19]. This benefits serial implementations such as AES-CBC, CCM and McOE-G. Furthermore, the throughput has been optimized a bit, which allows for better performance in parallel. Table 3 gives an overview of the latencies and inverse throughputs measured on our test machine (Core i5-4300U), which matches the numbers provided by Intel.

Table 3: Experimental latency (L) and inverse throughput (T^{-1}) of AES-NI and `pclmulqdq` instructions on Intel's Haswell microarchitecture. The data was obtained using the test suite of Fog [14].

Instruction	L	T^{-1}	Instruction	L	T^{-1}
`aesenc`	7	1	`aesimc`	14	2
`aesdec`	7	1	`aeskeygenassist`	10	8
`aesenclast`	7	1	`pclmulqdq`	7	2
`aesdeclast`	7	1			

4.2 Improvements for Multiplication in $GF(2^{128})$

The `pclmulqdq` instruction was introduced by Intel along with the AES-NI instructions [18], but is not part of AES-NI itself. The instruction takes two 128-bit inputs and a byte input `imm8`, and performs carry-less multiplication of a combination of one 64-bit half of each operand. The choice of halves of the two operands to be multiplied is determined by the value of bits 4 and 0 of `imm8`.

Most AE modes using multiplication in a finite field, employed in protocols in practice, use block lengths of 128 bits. As a consequence the finite fields multiplications are in $GF(2^{128})$. As the particular choice of finite field does influence the security proofs, modes use the tried-and-true GCM finite field.

For our performance study, we have used two different implementations of finite field multiplication (`gfmul`). The first implementation, which we refer to as the *classical method*, was introduced in Intel's white paper [18]. It applies `pclmulqdq` three times in a carry-less Karatsuba multiplication followed by modular reduction. The second implementation variant, which we refer to as the *Haswell-optimized method*, was proposed by Gueron [17] with the goal of leveraging the improved `pclmulqdq` performance on Haswell to trade many shifts and XORs for one more multiplication. This is motivated by the improvements in both latency (7 versus 14 cycles) and inverse throughput (2 versus 8 cycles) on Haswell [19].

In modes where the output of a multiplication over $GF(2^{128})$ is not directly used, other than as a part of a chain combined using addition, the aggregated reduction by Jankowski and Laurent [24] can be used to gain speed-ups. This method uses the inductive definitions of chaining values combined with the distributivity law for the finite field to postpone modular reduction at the cost of storing powers of an operand. Among the modes we benchmark in this work, the aggregated reduction method is applicable only to GCM. We therefore use aggregated reduction here, but apply the general `gfmul` implementations to the other modes.

4.3 Classical Versus Haswell $GF(2^{128})$ Multiplication

Here we compare the classical and Haswell-optimized methods of multiplication in $GF(2^{128})$ described in Section 4.2. We compare the performance of those AE modes that use full $GF(2^{128})$ multiplications (as opposed to aggregated reduction): McOE-G, COBRA and POET, when instantiated using the two different multiplication algorithms. Figure 2 shows that when processing a single message, the classical implementation of `gfmul` performs better than the Haswell-optimized method, while the situation is reversed when processing multiple messages in parallel.

Given the speed-up of `pclmulqdq` on Haswell, this may seem somewhat counter-intuitive at first. We observe, however, that McOE-G, COBRA and POET basically make sequential use of multiplications, which precludes utilizing the pipeline for single message implementations. In this case, the still substantial latency of `pclmulqdq` is enough to offset the gains by replacing several other instructions for the reduction. This is different in the multiple message case, where the availability of independent data allows our implementations to make more efficient use of the pipeline, leading to superior results over the classical multiplication method.

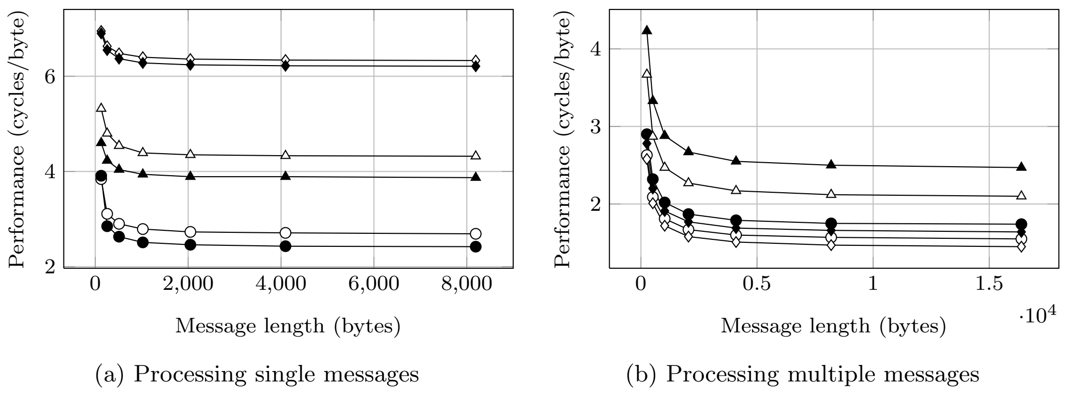


Fig. 2: Performance of McOE-G (diamond mark), COBRA (circle mark) and POET (triangle mark) with single messages (left) and 8 multiple messages (right). For both plots, the filled marks are performances using the *classical* `gfmul` implementation and the hollow marks are using the *Haswell-optimized* `gfmul` implementation.

5 Results

In this section we present the results of our performance study of the eight AE modes of operation considered in this paper.

5.1 The Setting

We provide optimized implementations of AE modes using the AES as the underlying block cipher and Intel's AES-NI instructions. We consider all combinations of messages sizes from 128 to 2048 bytes, single messages, and from two to 16 multiple messages.

All measurements were taken on a single core of an Intel Core i5-4300U CPU at 1900 MHz. For each combination of parameters, the performance was determined as the median of 91 averaged timings of 200 measurements each. This method has also been used by Krovetz and Rogaway in their benchmarking of authenticated encryption modes in [26].

For our performance measurements, we are interested in the performance of the various AE modes during their *bulk processing* of message blocks, i.e. during the encryption phase. To that end, we *do not* measure the time spent on processing associated data. As some schemes can have a significant overhead when computing authentication tags for short messages, we *do* include this phase in the measurements as well.

5.2 Performance Measurements

Table 4 lists the results of the performance measurements of the various AE modes for different message lengths in the single and multiple message setting. In the multiple message scenario, the performance is shown for the optimal number of multiple messages processed in parallel for each message length and algorithm.

In order to better evaluate the performance speed-ups (or reductions) obtained in the multiple message setting as opposed to processing single messages, the ratios of single to multiple message processing is shown in Table 5 for each of the parameters. Note that

Table 4: Performance comparison (in cycles/byte) of AE modes on various message lengths in the single and multiple message scenarios. For multiple messages, data is shown for the optimal level of parallelism for each message length and mode.

(a) Performance of nonce-based AE modes.

	Message length (bytes)				
Mode	128	256	512	1024	2048
	single message				
CCM	4.75	4.61	4.56	4.54	4.53
GCM	1.86	1.43	1.19	1.07	1.01
OCB3	1.94	1.27	0.94	0.77	0.72
OTR	2.64	1.19	1.00	0.90	0.86
	multiple messages				
CCM	1.37	1.29	1.25	1.23	1.21
GCM	1.61	1.53	1.49	1.47	1.46
OCB3	1.42	1.03	**0.84**	**0.74**	**0.69**
OTR	**1.14**	**0.96**	0.88	0.83	0.82

(b) Performance of nonce-misuse resistant AE modes.

	Message length (bytes)				
Mode	128	256	512	1024	2048
	single message				
McOE-G	6.90	6.55	6.37	6.28	6.24
COPA	3.00	2.34	2.02	1.85	1.76
COBRA	3.91	2.85	2.63	2.52	2.46
POET	4.61	4.24	4.13	4.02	3.92
	multiple messages				
McOE-G	1.70	1.56	1.49	1.46	1.44
COPA	**1.44**	**1.36**	**1.31**	**1.29**	**1.29**
COBRA	1.81	1.67	1.60	1.57	1.55
POET	2.37	2.27	2.17	2.12	2.10

numbers greater than 1 indicate a speed-up for multiple messages, while values smaller than 1 indicate a reduction in performance in comparison to the single message case.

Table 5: Ratios of single to multiple message performance of AE modes for various message lengths. This ratio describes the speed-up obtained by processing multiple messages of a certain length in parallel.

(a) Speed-up of multiple message processing in nonce-based AE modes.

	Message length (bytes)				
Mode	128	256	512	1024	2048
CCM	×3.47	×3.57	×3.65	×3.69	×3.74
GCM	×1.16	×0.93	×0.80	×0.73	×0.69
OCB3	×1.37	×1.23	×1.12	×1.04	×1.04
OTR	×2.32	×1.24	×1.14	×1.08	×1.05

(b) Speed-up of multiple message processing in nonce-misuse resistant AE modes.

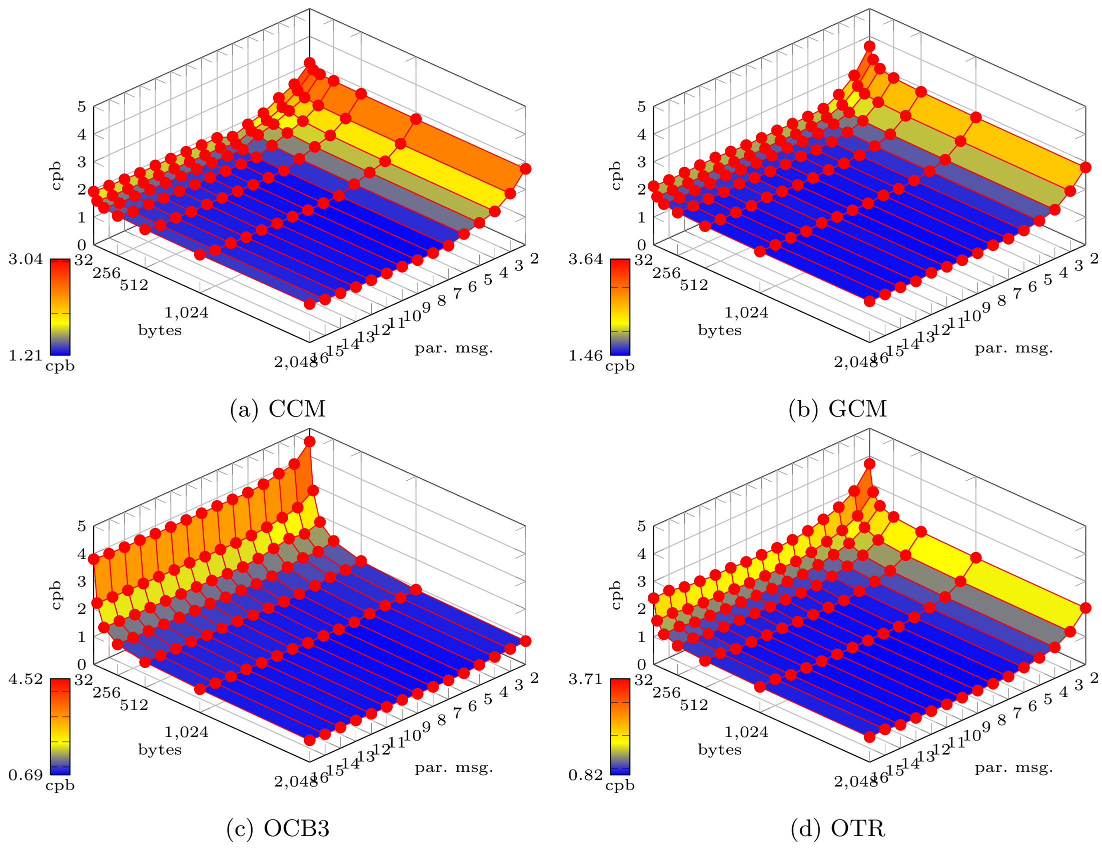
	Message length (bytes)				
Mode	128	256	512	1024	2048
McOE-G	×4.06	×4.20	×4.28	×4.30	×4.33
COPA	×2.08	×1.72	×1.54	×1.43	×1.36
COBRA	×2.16	×1.71	×1.64	×1.61	×1.59
POET	×1.95	×1.87	×1.90	×1.90	×1.87

The dependency of the performance in the multiple message case on the individual parameters is further detailed in Figs. 3 and 4. The horizontal lines in the color key of both figures indicate the integer values in the interval.

We further summarize the best-case performance of the various modes on long messages in Table 6.

5.3 Discussion

From Table 4, it becomes apparent, that for each message length, the optimum choice in terms of performance is a mode in the multiple message setting. In the nonce-based setting, for everything but very short messages, OCB3 performs best, with slight advantages over its own single message performance. For short messages (128 and 256 bytes), OTR outperforms OCB3 in the multiple message setting. For the nonce-misuse resistant modes, COPA with multiple messages consistently provides the best performance over all message lengths.



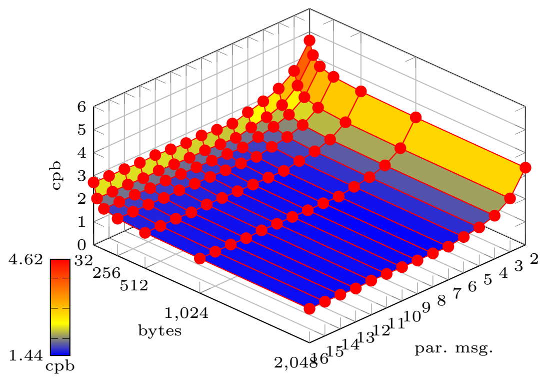
(a) CCM (b) GCM (c) OCB3 (d) OTR

Fig. 3: Performance of nonce-based AE modes of operation in the multiple-message setting.

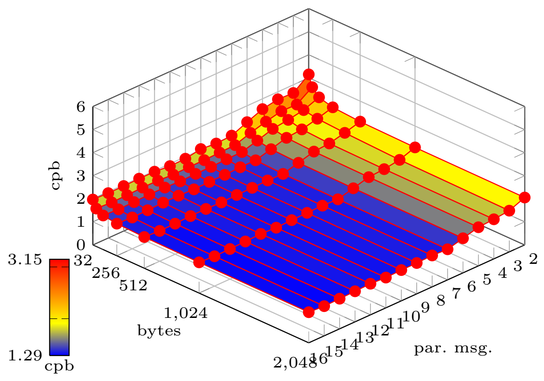
Considering the processing of a long single message, in the nonce-based setting, OCB3 with its 0.63 cpb is the mode of choice in all cases except for the short messages of about 128 bytes where GCM is slightly faster with 1.86 cpb and of about 256 bytes where OTR is better with 1.19 cpb. In the nonce-misuse resistant setting for single messages, COPA is clearly outperforming all other modes with 1.70 cpb, followed by COBRA with 2.42 cpb. The relation does not change for shorter messages of 128 bytes where COPA is still the mode of choice with 3.00 cpb.

The landscape changes significantly when multiple messages are processed in parallel. For almost all AE modes considered, we obtain a consistent speed-up when processing multiple messages at the same time, see Table 5. Notably, among the nonce-based modes, CCM gets by factor 3.5 faster and its performance is about 1.2 cpb which is close to that of GCM (the latter, however, possessing classes of weak keys), with OCB3 still performing at only 0.69 cpb. Among the nonce-misuse resistant modes, McOE-G receives a speed-up by factor 4 and its performance is about 1.44 cpb, which is faster than COBRA with its 1.55 cpb but slower than for COPA with 1.29 cpb.

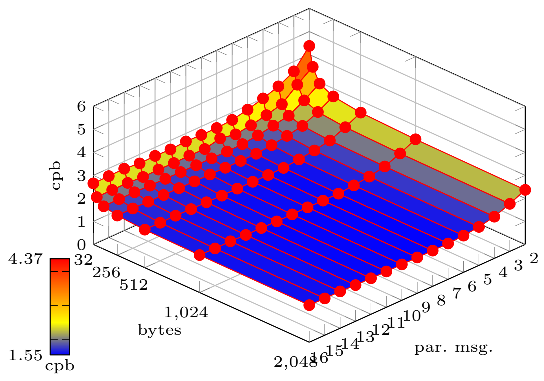
We also note from Table 6 that most modes achieve their best speed-up in the multiple messages scenario for around 7-8 multiple messages, with GCM and COPA being the exceptions at 13 and 15, respectively. A closer inspection of Figures 3 and 4 reveals that the GCM performance is actually quite stable from 7 till 13 multiple messages, starting from when the increased pressure on the available AVX registers starts to influence the results. COPA is a different case, though, with its two block cipher calls per message block explaining its preference for about twice as many available messages than the pipeline length.



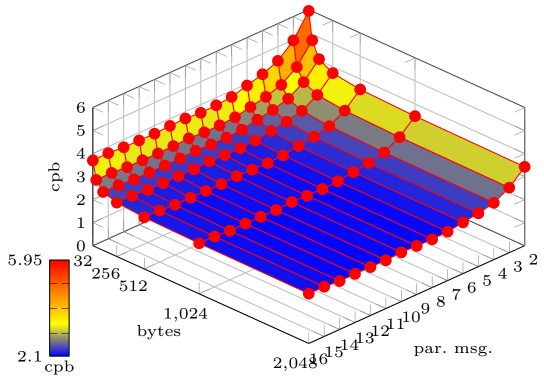
(a) McOE-G



(b) COPA



(c) COBRA



(d) POET

Fig. 4: Performance of nonce-misuse resistant AE modes of operation in the multiple-message setting.

6 Conclusion

In this paper, we have discussed the performance of (authenticated) modes of operation for block ciphers using the AES on Intel's recent Haswell architecture.

As a general technique to speed up both inherently sequential modes and to the scenario of having many but shorter messages, we propose the scheduling of multiple messages in parallel *already in the implementation of the algorithm itself*. This leads to significant speed-ups for serial modes, but also to considerable improvements for parallelizable modes.

We then apply this to AES-CBC encryption and illustrate the potential in this approach by obtaining a speed-up of around factor 6.5 for longer messages of 2048 bytes, leading to a performance of 0.56 cpb on a single core.

Providing the first optimized AES-NI implementations of the recent AE modes OTR, McOE-G, COBRA, and POET, we perform a comprehensive performance study of both nonce-based and nonce-misuse resistant modes on a wide range of parameters. Our study indicates that processing multiple messages is almost always beneficial, even for parallelizable modes. When processing single messages, OCB3 achieves the best performance for short messages among the nonce-based modes we analyze; among the nonce-misuse resistant modes, COPA performs best for all message lengths.

References

1. Cryptographic Competitions. Accessed on February 17, 2014., February.
2. Farzaneh Abed, Scott Fluhrer, Christian Forler, Eik List, Stefan Lucks, David McGrew, and Jakob Wenzel. Pipelineable On-Line Encryption. In *FSE*, 2014.

Table 6: Performance of block cipher and AE modes on long messages (2048 bytes) in single and multiple message settings.

Mode	Single msg.	Multiple msg. (# of msgs.)
ECB	0.55	0.54 (8)
CTR	0.64	0.65 (8)
CBC	3.80	0.56 (8)
CCM	4.53	1.21 (8)
GCM	1.01	1.46 (13)
OCB3	0.72	0.69 (7)
OTR	0.86	0.82 (8)
McOE-G	6.24	1.44 (7)
COPA	1.76	1.29 (15)
COBRA	2.46	1.55 (8)
POET	3.93	2.10 (8)

3. National Security Agency. NSA Suite B Cryptography. Accessed on February 17, 2014. `http://www.nsa.gov/ia/programs/suiteb_cryptography/`, February 2014.
4. Kahraman Akdemir, Martin Dixon, Wajdi Feghali, Patrick Fay, Vinodh Gopal, Jim Guilford, Erdinc Ozturk, Gil Wolrich, and Ronen Zohar. *Breakthrough AES Performance with Intel AES New Instructions.* Intel Corporation, 2010.
5. Elena Andreeva, Begl Bilgin, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. APE: Authenticated Permutation-Based Encryption for Lightweight Cryptography. In *FSE*, 2014. to appear.
6. Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Elmar Tischhauser, and Kan Yasuda. Parallelizable and Authenticated Online Ciphers. In *ASIACRYPT (1)*, pages 424–443, 2013.
7. Elena Andreeva, Atul Luykx, Bart Mennink, and Kan Yasuda. COBRA: A Parallelizable Authenticated Online Cipher Without Block Cipher Inverse. In *FSE*, 2014.
8. Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications. In *Selected Areas in Cryptography*, pages 320–337, 2011.
9. Begl Bilgin, Andrey Bogdanov, Miroslav Knezevic, Florian Mendel, and Qingju Wang. Fides: Lightweight Authenticated Cipher with Side-Channel Resistance for Constrained Hardware. In *CHES*, pages 142–158, 2013.
10. Andrey Bogdanov, Florian Mendel, Francesco Regazzoni, Vincent Rijmen, and Elmar Tischhauser. ALE: AES-based Lightweight Authenticated Encryption. In *FSE*, 2013.
11. T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176.
12. Ewan Fleischmann, Christian Forler, and Stefan Lucks. McOE: A Family of Almost Foolproof On-Line Authenticated Encryption Schemes. In *FSE*, pages 196–215, 2012.
13. Ewan Fleischmann, Christian Forler, Stefan Lucks, and Jakob Wenzel. McOE: A Family of Almost Foolproof On-Line Authenticated Encryption Schemes. Cryptology ePrint Archive, Report 2011/644, 2011. `http://eprint.iacr.org/`.
14. Agner Fog. Software Optimization Resources. Accessed on February 17, 2014. `http://www.agner.org/optimize/`, February 2014.
15. Christian Forler, Stefan Lucks, David McGrew, and Jakob Wenzel. Presented at DIAC 2012. Hash-CFB. July 2012.
16. Shay Gueron. *Intel Advanced Encryption Standard (AES) New Instructions Set.* Intel Corporation, 2010.
17. Shay Gueron. Aes-gcm software performance on the current high end cpus as a performance baseline for caesar. In *DIAC 2013: Directions in Authenticated Ciphers*, 2013.
18. Shay Gueron and Michael E. Kounavis. *Intel Carry-Less Multiplication Instruction and its Usage for Computing the GCM Mode.* Intel Corporation, 2010.
19. Sean Gulley and Vinodh Gopal. *Haswell Cryptographic Performance.* Intel Corporation, 2013.
20. Brent Hollingsworth. *New "Bulldozer" and "Piledriver" Instructions.* Advanced Micro Devices, Inc., 2012.
21. R. Housley. Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP). RFC 4309 (Proposed Standard), December 2005.
22. Steven Iveson. IPSec Bandwidth Overhead Using AES. Accessed on February 17, 2014. `http://packetpushers.net/ipsec-bandwidth-overhead-using-aes/`, October 2013.

23. Tetsu Iwata, Kazuhiko Minematsu, Jian Guo, and Sumio Morioka. CLOC: Authenticated Encryption for Short Input. In *FSE*, 2014. to appear.
24. Krzysztof Jankowski and Pierre Laurent. Packed AES-GCM Algorithm Suitable for AES/PCLMULQDQ Instructions. pages 135–138, 2011.
25. Wolfgang John and Sven Tafvelin. Analysis of internet backbone traffic and header anomalies observed. In *Internet Measurement Comference*, pages 111–116, 2007.
26. Ted Krovetz and Phillip Rogaway. The Software Performance of Authenticated-Encryption Modes. In *FSE*, pages 306–327, 2011.
27. David A. McGrew and John Viega. The Galois/Counter Mode of Operation (GCM).
28. David A. McGrew and John Viega. The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In *INDOCRYPT*, pages 343–355, 2004.
29. Kazuhiko Minematsu. Parallelizable Authenticated Encryption from Functions. Cryptology ePrint Archive, Report 2013/628, 2013. `http://eprint.iacr.org/`.
30. David Murray and Terry Koziniec. The state of enterprise network traffic in 2012. In *Communications (APCC), 2012 18th Asia-Pacific Conference on*, pages 179–184. IEEE, 2012.
31. Kostas Pentikousis and Hussein G. Badr. Quantifying the deployment of TCP options - a comparative study. pages 647–649, 2004.
32. Phillip Rogaway. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In *ASIACRYPT*, pages 16–31, 2004.
33. Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: A Block-cipher Mode of Operation for Efficient Authenticated Encryption. In *ACM Conference on Computer and Communications Security*, pages 196–205, 2001.
34. Doug Whiting, Russ Housley, and Niels Ferguson. Counter with CBC-MAC (CCM), 2003.
35. Hongjun Wu and Bart Preneel. AEGIS: A Fast Authenticated Encryption Algorithm. page 695, 2013.

A Additional Performance Data

Table 7: Performance of authenticated encryption modes from short to long messages (single message case).

	Message length (bytes)						
	128	256	512	1024	2048	4096	8192
CCM	4.75	4.61	4.56	4.54	4.53	4.53	4.53
GCM	**1.86**	1.43	1.19	1.07	1.01	0.98	0.97
OCB3	1.94	1.27	**0.93**	**0.78**	**0.70**	**0.65**	**0.63**
OTR	2.64	**1.19**	1.00	0.90	0.86	0.83	0.82
COBRA	3.91	2.85	2.63	2.52	2.46	2.43	2.42
COPA	**3.00**	**2.34**	**2.02**	**1.85**	**1.76**	**1.72**	**1.70**
McOE-G	6.90	6.55	6.37	6.28	6.24	6.22	6.21
POET	4.61	4.24	4.13	4.02	3.92	3.88	3.87

Table 8: Normalized performance comparison of AE modes.

(a) Normalized performance (in cpb) of nonce-based AE modes, relative to OCB3 for each message length.

	Message length (bytes)				
Mode	128	256	512	1024	2048
			single message		
CCM	×3.35	×4.48	×5.43	×6.14	×6.57
GCM	×1.31	×1.39	×1.42	×1.45	×1.46
OCB3	×1.37	×1.23	×1.12	×1.04	×1.04
OTR	×1.86	×1.16	×1.19	×1.22	×1.25
			multiple messages		
CCM	×0.96	×1.25	×1.49	×1.66	×1.75
GCM	×1.13	×1.49	×1.77	×1.99	×2.12
OCB3	×1.00	×1.00	×1.00	×1.00	×1.00
OTR	×0.80	×0.93	×1.05	×1.12	×1.19

(b) Normalized performance (in cpb) of nonc-misuse resistant AE modes, relative to COPA for each message length.

	Message length (bytes)				
Mode	128	256	512	1024	2048
			single message		
McOE-G	×4.79	×4.82	×4.86	×4.87	×4.84
COPA	×2.08	×1.72	×1.54	×1.43	×1.36
COBRA	×2.72	×2.10	×2.01	×1.95	×1.91
POET	×3.20	×3.12	×3.15	×3.12	×3.04
			multiple messages		
McOE-G	×1.18	×1.15	×1.14	×1.13	×1.12
COPA	×1.00	×1.00	×1.00	×1.00	×1.00
COBRA	×1.26	×1.23	×1.22	×1.22	×1.20
POET	×1.65	×1.67	×1.66	×1.64	×1.63