Software Benchmarking of the 2nd round CAESAR Candidates

Ralph Ankele[1] and Robin Ankele[2]

[1] Information Security Group, Royal Holloway, University of London
McCrea Building, Egham TW20 0EX, UK
`ralph.ankele.2015@rhul.ac.uk`

[2] Department of Computer Science, University of Oxford
Wolfson Building, Oxford OX1 3QD, UK
`robin.ankele@cs.ox.ac.uk`

Abstract. The software performance of cryptographic schemes is an important factor in the decision to include such a scheme in real-world protocols like TLS, SSH or IPsec. In this paper, we develop a benchmarking framework to perform software performance measurements on authenticated encryption schemes. In particular, we apply our framework to independently benchmark the 29 remaining 2nd round candidates of the CAESAR competition. Many of these candidates have multiple parameter choices, or deploy software optimised versions raising our total number of benchmarked implementations to 207. We illustrate our results in various diagrams and hope that our contribution helps developers to find an appropriate cipher in their selection choice.

Keywords: Software benchmarks, CAESAR, 2nd round candidates, real-worlds usecases, tls, ssh, optimisations, aes-ni, sse, avx, neon

1 Introduction

An authenticated encryption (AE) scheme provides confidentiality, integrity and authenticity simultaneously. These goals are typically achieved by combining separate cryptographic primitives, an encryption scheme to ensure confidentiality and a message authentication code for integrity and authenticity. However, the generic composition, the combination of a confidentiality mode with an authentication mode, may lead to implementation errors and is often not efficient (*e.g.* the input stream has to be processed twice, for the encryption scheme and the message authentication code). Recently, some practical attacks [14], [41] have confirmed the vulnerability of these AE schemes. In 2013 the CAESAR [8] competition was announced to overcome these attacks and to achieve fast and efficient AE schemes and should be suitable for widespread adoption. In March 2014, 57 first round candidates were submitted to the CAESAR competition. Abed *et al.* [4] provide a classification and a general overview of the first round candidates. In July 2015, the second round candidates were announced. The deadline for the announcement of the third round candidates is expected in end July 2016 that requires the CAESAR committee to make a selection decision which candidates will be included in the third round.

Beside of the security guarantees of the authenticated encryption mode, the software and hardware performance of the candidate becomes more and more important. One of the requirements imposed on all CAESAR submissions was that they should demonstrate the potential to be superior to AES-GCM in at least one significant aspect. One of those aspects that is particularly significant for the usage in widely deployed protocols such as TLS, SSH and IPsec is software performance. Therefore, we have developed a benchmarking framework for automated and accurate software performance measurements and applied it to benchmark the remaining 29 second round candidates. Many of these candidates have multiple parameter choices, extending the number to 125 implementations. Overall, we have considered several software performance optimised implementations, raising the total number of benchmarked implementations to 207.

During our benchmarks, we perform and illustrate our measurements on a continuous function (with message and associated data sizes from 0 to 2048 bytes, in 128 byte steps) and on some fixed length sizes, relating to real-world usecases — 1 byte (one keystroke , SSH), 16 bytes (small payload), 557 bytes (average IP packet size), 1.5 kB (ethernet MTU size, TLS), 16 kB (max. TCP packet size) and 1 MB (*e.g.* file upload).

Related Work. In order to evaluate the software performance of the CAESAR candidates, Bernstein provides the SUPERCOP[⋆⋆⋆] [9] framework, which is a general purpose framework for the evaluation of the software performance of cryptographic software. Saarinen presented the BRUTUS[†] [38] framework, in order to test the CAESAR candidates for cryptographic weaknesses. Additionally to the software performance of the candidates, the hardware performance is also an important factor in the selection of the final portfolio. Homsirikamol *et al.* [23] present a hardware API to perform hardware benchmarks of the CAESAR candidates.

Contribution and Outline. In this paper, we present a complementary benchmarking framework to perform automatic and accurate measurements on authenticated encryption functions. We apply this framework to perform independent software benchmarks on the 2nd round CAESAR candidates and illustrate the results in various diagrams.

This paper is organised as follows. In Section 2, we introduce the 2nd round candidates of the CAESAR competition and classify them regarding their underlying cryptographic primitive. Section 3, presents our benchmarking framework and the measurement settings. In Section 4, we give an overview and discuss the software optimisation implemented in the 2nd round candidates. Finally, in Section 5 we present the results of the benchmarking and Section 6 concludes the paper.

[⋆⋆⋆] `https://bench.cr.yp.to/supercop.html`

[†] `https://github.com/mjosaarinen/brutus`

2 Classification of the 2^{nd} round CAESAR Candidates

In the CAESAR [8] call for submissions the choice of underlying primitive and type of cipher was open for the designers. After 57 submissions for the first round, 29 submissions have been selected for the second round. Within these remaining candidates there are 15 based on block ciphers, 8 based on sponge constructions, 4 based on stream ciphers, 2 based permutations and 1 based on a compression function. Table 1 gives an overview of the 2^{nd} round candidates of the CAESAR competition sorted by type.

Some of the candidates are *online*. An online cipher is defined such that the encryption of message block M_i is only depending on already processed message blocks $M_1, \ldots, M_{i-1}$. Furthermore, an inverse-free scheme indicates if the inverse of an underlying primitive is needed (*i.e.* for the purpose of decryption of the underlying block cipher). An authenticated encryption scheme is nonce-misuse resistant, if the cipher is still secure under the repetition of nonces. Moreover, we can distinguish between nonce misuse resistance for online and offline ciphers. On the one hand, offline ciphers security can be claimed if the repetition of the ciphers leak only the ability to see a repeated message. On the other hand, for online ciphers security can be claimed, if nonces are repeated only the common prefix of the messages can be observed.

3 Benchmarking Framework

Similar to the SUPERCOP [9] and BRUTUS [38] frameworks, we have developed a benchmarking framework to facilitate automatic software performance evaluations for authenticated encryption functions. Using our toolkit, one can perform independent and accurate software performance measurements (*i.e.* timing measurements), verify the implementation of the cipher under test, generate and illustrate statistical information about the software performance, and compare different parameters and optimisations of a cipher among its different optimisations or to other ciphers under test. Nevertheless, our benchmarking suite is currently optimised to perform measurements of authenticated encryption functions it remains open to be extended to perform benchmarking of other functionality and include other types of software tests.

Generally, our benchmarking framework provides the following benefits compared to the SUPERCOP and BRUTUS framework.

- ◇ Very simple with only focus on comparison of AE schemes
- ◇ Due to its simplicity open for extension of other software performance tests
- ◇ Optimised TSC instruction (*i.e.* `rdtscp`) for accurate timing measurements
- ◇ Reduction of noise in the measurement using single user mode, averaging and median
- ◇ Benchmarking, verification, illustration and comparison of results

Using our framework, we built and benchmarked the 2^{nd} round candidates (see Table 1) of the ongoing CAESAR competition. By July 2016, this includes

Table 1: Overview of the 2nd round CAESAR candidates.‡Possible types of schemes are: BC: block cipher based schemes, SC: stream cipher based schemes, Sponge: scheme based on Sponge construction (duplex or otherwise), P: permutation based scheme and CF: scheme based on compression function. Underlying primitives are: AES: when the full AES-128 is used, AES[r]: when a round reduced version of AES is used, other dedicated primitive or constructions: e.g. SHA2, SPN, LFSR. Entries for nonce misuse resistance are: None: when no security is claimed by the designers if nonces are repeated, Max: when for offline ciphers the repetition of nonces only leak the ability to see a repeated message, LCP: for online ciphers, all an adversary can observe is the longest common prefix of messages for repeated nonces.

Name	Type	Primitive	Parallel Enc. / Dec.	Online Enc. / Dec.	Inverse-free	Security proof	Nonce misuse resistant	Reference
AEGIS	BC	AES[1]	✓/ -	✓/✓	-	-	None	[45]
AES-COPA	BC	AES	partly/partly	✓/✓	-	✓	LCP	[6]
AES-JAMBU	BC	AES, Simon	- / -	✓/✓	✓	-	LCP	[44]
AES-OTR	BC	AES	✓/✓	✓/✓	✓	✓	None	[33]
AEZ	BC	AES[4]	✓/✓	- / -	✓	✓	Max	[22]
CLOC	BC	AES, TWINE	- / -	✓/✓	-	✓	None	[25]
Deoxys	BC	AES	✓/✓	✓/✓	-	✓	LCP	[27]
ELmD	BC	AES	partly/partly	✓/✓	-	✓	LCP	[17]
Joltik	BC	Joltik-BC	✓/✓	✓/✓	-	✓	LCP	[28]
OCB	BC	AES	✓/✓	✓/✓	-	✓	None	[31]
POET	BC	AES	partly/partly	✓/✓	✓	✓	LCP	[3]
SCREAM	BC	SPN	✓/✓	✓/✓	-	-	None	[20]
SHELL	BC	AES	partly/partly	✓/✓	-	✓	None	[42]
SILC	BC	AES, Present, LED	- /-	✓/✓	✓	✓	None	[26]
Tiaoxin	BC	AES[1]	✓/✓	✓/✓	✓	-	None	[35]
OMD	CF	SHA2	- / -	✓/✓	-	✓	None	[16]
Minalpher	P	SPN	✓/✓	✓/✓		✓		[39]
PAEQ	P	AESQ	✓/✓	✓/✓	✓	✓		[12]
ACORN	SC	LFSR	✓/✓	✓/✓	✓	-	None	[43]
HS1-SIV	SC	ChaCha/Poly1305	- / -	- / -	✓	✓	Max	[29]
MORUS	SC	LRX	- / -	✓/✓	✓	-	None	[24]
TriviA-ck	SC	Trivium	partly/partly	- / -	✓	✓	None	[15]
Ascon	Sponge	SPN	- / -	✓/✓	✓	✓		[18]
ICEPOLE	Sponge	Keccack-like	✓/ ✓	✓/✓	✓	✓		[34]
Ketje	Sponge	Keccack-f	- / -	✓/✓	✓	✓	None	[10]
Keyak	Sponge	Keccack-f	✓/ ✓	✓/✓	✓	✓	None	[11]
NORX	Sponge	LRX	✓/✓	✓/✓	✓	✓	None	[7]
π-cipher	Sponge	ARX	✓/✓	✓/✓	✓	-	None	[19]
PRIMATEs	Sponge	SPN	- / -	✓/✓	✓	✓	LCP	[5]
STRIBOB	Sponge	Streebog	- / -	✓/✓	✓	✓	None	[36]

the implementations of 29 candidates. Many of these candidates have multiple parameter choices, extending the number to 125 implementations. Overall, we have considered several software performance optimised implementations, raising the total number of benchmarked implementations to 207[§].

3.1 Measurement Setup

All measurements were taken on a 64-bit Sandy Bridge 2.3GHz dual-core Intel Core i5-2415M processor. The underlying hardware was a Mac Book Pro Early 2011 running OS X 10.11 (El Capitan) in single user mode. The implementations under test were built using the clang compiler version 6.1.0 (clang-602.0.53). Additionally, we enabled the following compiler flags for compile time optimisation of the underlying source code: `-Ofast -fno-stack-protector -march=native`. In order to get rid of the noise(*e.g.* context switches) during the measurement and to get a clean and stable result in an uninterrupted environment, we choose to boot in the single user mode, which just starts the core operating system and provides a `sh` command interface, without starting other core elements of the OS X operating system (*e.g.* network services, multi-user support, graphical user interface) that may interfering with our benchmarking. Furthermore, we used the same method as described in the paper from Krovetz and Rogaway [30]. In their approach, they determine the performance of the underlying function as the median of 91 averaged timings of 200 measurements each.

3.2 High Resolution Methods for CPU Timing Information

In general, there are several ways to determine the software performance of a function under test. Each operating system offers several methods of accurate timing sources to be used to determine the runtime (*e.g.* latency) or throughput of a function. Using multi-core processors, one must be aware of handling hyper-threading and over-clocking of the CPU during their benchmarking, to avoid distorted or falsified measurements.

For the performance of high precision timing measurements on Windows platforms, one could use the HPET (i.e. High Precision Event Timer) or the QueryPerformanceCounter. On Unix based platforms one can use the posix conform `time()` and `clock()` functions, as used in the BRUTUS framework. However, a much more accurate timing source on x86 processors is the Timer Stamp Counter (TSC) as used in the SUPERCOP framework. The TSC is a 64 bit register and counts the number of cycles since the last reset of the machine. Using the `RDTSC` assembler instruction, one can read out the current TSC value and use it for accurate timing measurements.

In newer processor hardware, Intel processors practice out-of-order execution, where instructions are not always performed in the order as they appear in the source code. This could lead to a skewed measurement result. To avoid such

[‡] Data obtained from: https://aezoo.compute.dtu.dk/doku.php

[§] Number of implementations supported by our benchmarking hardware.

Table 2: Real-world use case settings for our benchmarking.

Message Size	Associated Data Size	Comments
1 byte	5 byte	one keystroke (*e.g.* SSH)
16 bytes	5 byte	small payload
557 byte	5 byte	average IP packet size[¶]
1.5 kB	5 byte	ethernet MTU, TLS
16 kB	5 byte	max TCP packet size
1 MB	5 byte	file upload

errors, the `CPUID` instruction, a serialisation instruction, can be inserted, which forces every preceding instruction to complete before the execution of the `RDTSC` instruction. In our framework, we make use of the `RDTSCP` [37] instruction, which is an optimised versions of the combination `CPUID` + `RDTSC`.

3.3 Authenticated Encryption Benchmarking Settings and Real-World Usecases

In our benchmarking, we implemented measurements on a continuous function (*i.e.* where we vary the message size and the associated data size) and over some fixed message and associated data sizes (*i.e.* representing some real-world usecases).

Overall, we consider message size from 0 to 1 MB and associated data size from 0 to 2 KB. The sizes of the key and nonce are fixed-size values, as determined by the CAESAR specification. In the case of the continuous function we consider message and associated data size from 0 to 2048 bytes in 128 byte steps. On the other hand, for the timing measurements with fixed message and associated data size, we consider the values in Table 2.

The associated data is fixed to 5 bytes[‖] for all measurements, which represents the typical size of the header of a TLS packet. Additionally, for better comparability, we represent the performance results of the authenticated encryption candidates in cycle per byte (cbp), which is a function of the throughput of the ciphers, instead of releasing the timings (*i.e.* latency) of the candidates.

4 Software Optimisations

Several ciphers are supported by software-optimised implementations. These optimisations are achieved by the usage of architecture-specific instructions and intrinsics. An intrinsic function is handled in a special manner by the compiler where the intrinsic function maps to some specific assembler instructions. Table 3 gives an overview of the mainly used instruction set extensions in the 2nd round candidates. Furthermore, in the following we give a short overview of

[‖] `http://netsekure.org/2010/03/tls-overhead`

Table 3: Most common optimisations for the CAESAR candidates

Instruction Set	Vendor	Microarchitecture	CAESAR Candidate
AES-NI	Intel	Westmere	OCB, AEGIS, CLOC, SILC, POET, JAMBU, AEZ, Deoxys, PAEQ, Tiaoxin
SSE	Intel	Pentium III	OCB, CLOC, Keyak, Morus, OMD, Scream, Stribob, Tiaoxin
AVX	Intel	Sandy Bridge	NORX, OMD
AVX2	Intel	Haswell	Keyak, Minalpher, Morus, NORX
NEON	ARM	ARMv7-A	OCB, NORX, Scream, Stribob

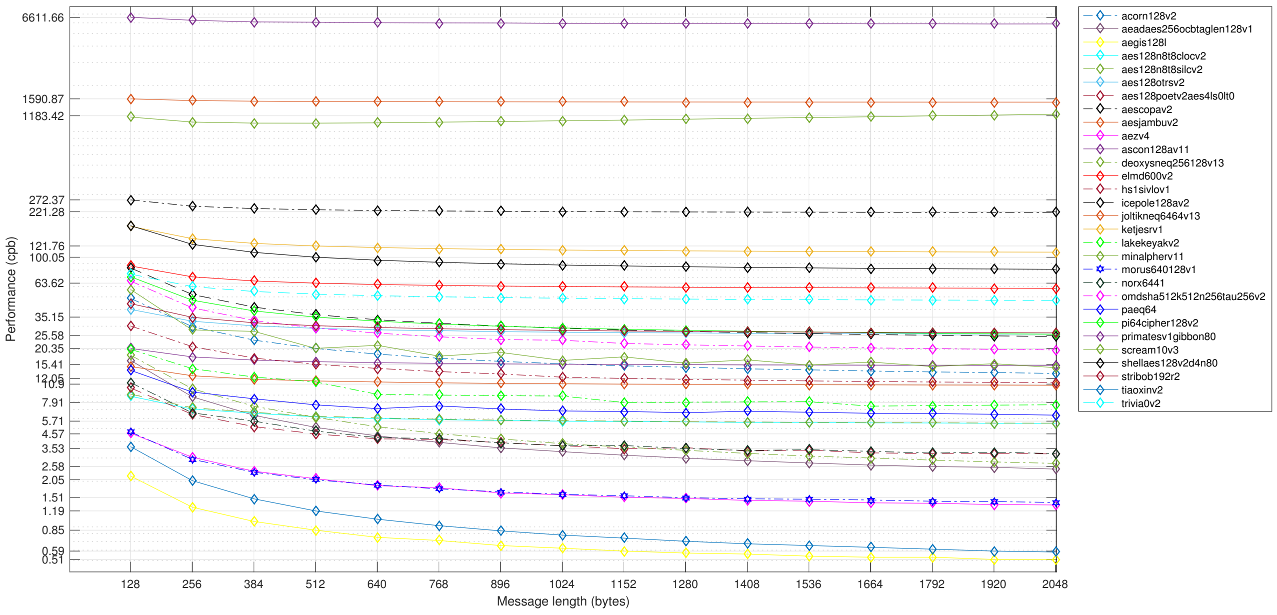
the most common used instruction set extensions to achieve software optimised algorithms.

AES-NI. As many CAESAR candidates are AES based, they can obtain an increase in software performance by using the AES New Instructions [21]. These set of instructions are available beginning with the 2010 Intel processors based on the 32nm microarchitecture Westmere. The instruction set consists of six instructions that offer full hardware support for AES enabling fast and secure encryption and decryption. The AES New Instructions increase the performance by more than an order of magnitude for parallel modes (*e.g.* CTR mode), and provide roughly 2-3 fold gains for non-parallelisable modes (*e.g.* CBC mode). On a Westmere microarchitecture the AES-NI instructions can run with approximately 1.3 cycles per byte in parallel mode of operation.

SSE. The Streaming SIMD Extensions [40] contain around 70 instructions for high performance optimised implementations. The SSE instructions are available since Pentium III and the latest update SSE4.2 is available with the Nehalem microarchitecture. The extensions provide up to 16 128-bit registers (xmm0-15) and provide vector-mode operations, enabling parallel execution of one operation on multiple data. The speedup that can be achieved with the SSE instructions depends on whether the data can be parallelised, as operations are used in a vector-mode where the same operation is executed on multiple data.

AVX. The Advanced Vector Extensions [32] were introduced with the Sandy Bridge microarchitecture and extend the SSE instructions with a new set of instructions and a new coding scheme. The AVX1 instructions extend the previous xmm0-15 128-bit registers to 16 256-bit registers (ymm0-15). Furthermore, memory alignments have been relaxed and three-operand non-destructive operations have been added. Previously, with two-operand commands the source operand was always overwritten by the instruction (*e.g.* A = A + B). With the new three-operand instructions (*e.g.* A = B + C) an additional `mov` instruction to secure the source-value is not necessary. Additionally, the AVX2 [13] instructions are an advancement of the AVX instructions supporting integer datatypes

Fig. 1: Comparison of the best implementation for each CAESAR candidate. The associated data is fixed to 0 bytes.



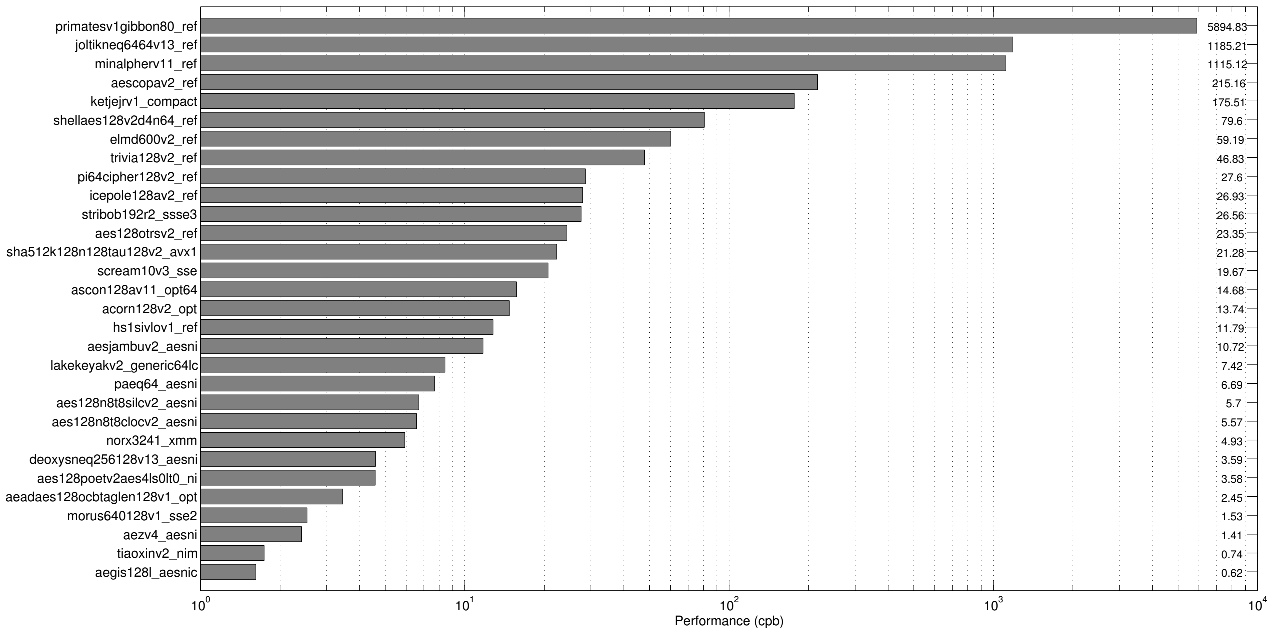
and adding vector shift operations. The new instructions were introduced with the Haswell microarchitecture.

NEON. The implementation of the Advanced SIMD extensions used in ARM processors is called NEON [1] and is available with the Cortex-A microarchitecture. ARM processors are used in many mobile devices such as tablets and mobile phones. The NEON instructions are 128-bit SIMD extensions providing a flexible and powerful acceleration as they perform packaged SIMD processing. Registers are therefore considered as vectors of elements with the same data type and perform the same operation on all lanes.

5 Results

We used our previously described benchmark framework to benchmark all 2nd round CAESAR candidates. Figure 1 shows a comparison of the best implementations for each CEASAR candidate. To demonstrate the impact of a CAESAR candidate in a real-world setting, we also made benchmarks in the following settings. Figure 2 compares the best implementations per CAESAR candidate for a common TLS setting. We chose the TLS setting, because we believe it is the most common usecase for authenticated encryption. Therefore, we set the associated

Fig. 2: Comparison of 2nd round CAESAR candidates in the TLS setting.



data length to 5 bytes[**] which represents the typical length of a TLS header. The message length is set to 1500 bytes, which corresponds to the MTU size of an ethernet frame. Moreover, we also made benchmarks for the SSH setting. In this setting, we fix the associated data length to 5 bytes and the message length to 1 byte (*i.e.* one key stroke). Figure 3 compares the best implementations per CAESAR candidate for the SSH setting. Finally, we created benchmarks for all candidates with increasing message/associated data lengths. Additional 3D plots are provided for the best implementation of each cipher, in order to illustrate the evolution in performance when the message length and the associated data length increase. We give detailed illustrations of these benchmarks in the appendices. The benchmarking framework and our scripts to visualise the results are available online[††] for verification purposes and to encourage other designers and researchers to use our framework for benchmarking their cryptographic schemes.

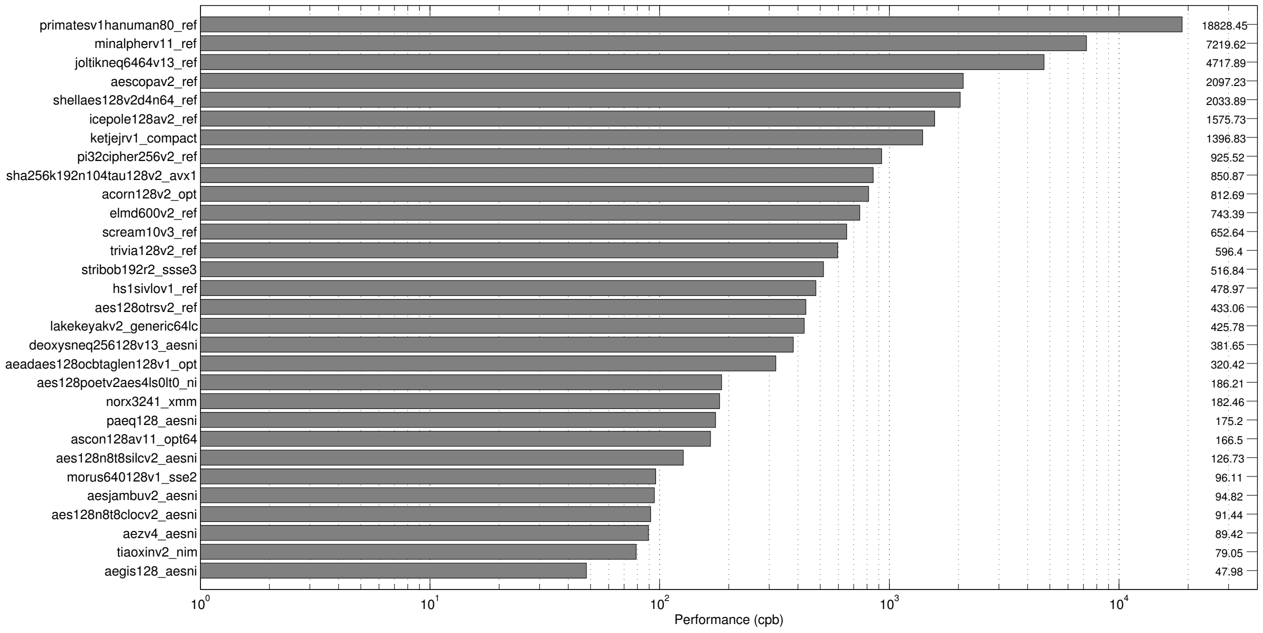
6 Conclusion

In this paper, we present a new benchmarking framework to perform software performance measurements of authenticated encryption schemes. We apply our

[**] `http://netsekure.org/2010/03/tls-overhead`

[††] `https://github.com/TheBananaMan/caesar_benchmarks_secondround`

Fig. 3: Comparison of 2nd round CAESAR candidates in the SSH setting.



framework to benchmark the 2nd round CAESAR candidates. Software performance is an important factor in real-world applications. Our benchmarks show that candidates supported by a software-optimised implementation achieve a huge speedup with regards to others, without an optimised implementation. The aim of the CAESAR competition is to provide a portfolio of authenticated encryption algorithms, including implementations optimised for software (and hardware), as in the eStream portfolio [2]. Currently, nearly two thirds of the candidates are supported by at least one software optimisation. Nevertheless, with our benchmarks we want to encourage designers to increase the number of optimised implementations.

Acknowledgements

We would like to thank the designers of the CAESAR candidates, which supported us with their software implementations. The first author is supported by the Marie Sklodowska-Curie ITN ECRYPT-NET grant (Project Reference 643161).

References

1. Neon, `http://www.arm.com/products/processors/technologies/neon.php`

2. estream: the ecrypt stream cipher project. http://www.ecrypt.eu.org/stream/ (2008)
3. Abed, F., Fluhrer, S.R., Foley, J., Forler, C., List, E., Lucks, S., McGrew, D., Wenzel, J.: The POET Family of On-Line Authenticated Encryption Schemes. https://competitions.cr.yp.to/round2/poetv20.pdf (2015)
4. Abed, F., Forler, C., Lucks, S.: General Overview of the Authenticated Schemes for the First Round of the CAESAR Competition. Cryptology ePrint Archive, Report 2014/792 (2014)
5. Andreeva, E., Bilgin, B., Bogdanov, A., Luykx, A., Mendel, F., Mennink, B., Mouha, N., Wang, Q., Yasuda, K.: PRIMATEs v1.02. https://competitions.cr.yp.to/round2/primatesv102.pdf (2014)
6. Andreeva, E., Bogdanov, A., Luykx, A., Mennink, B., Tischhauser, E., Yasuda, K.: AES-COPA v.2. https://competitions.cr.yp.to/round2/aescopav2.pdf (2015)
7. Aumasson, J.P., Jovanovic, P., Neves, S.: NORX v2.0. https://competitions.cr.yp.to/round2/norxv20.pdf (2015)
8. Bernstein, D.J.: Caesar: Competition for authenticated encryption: Security, applicability, and robustness. https://competitions.cr.yp.to/caesar.html (2014)
9. Bernstein, D.J.: Supercop. https://bench.cr.yp.to/supercop.html (2016)
10. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., Van Keer, R.: CAESAR submission: KETJE v1. https://competitions.cr.yp.to/round1/ketjev1.pdf (2014)
11. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., Van Keer, R.: CAESAR submission: KEYAK v2. https://competitions.cr.yp.to/round2/keyakv21.pdf (2015)
12. Biryukov, A., Khovratovich, D.: PAEQ v1. https://competitions.cr.yp.to/round1/paeqv1.pdf (2014)
13. Buxton, M.: Intel® advanced vector extensions programming reference. https://software.intel.com/en-us/blogs/2011/06/13/haswell-new-instruction-descriptions-now-available (2011)
14. Canvel, B., Hiltgen, A., Vaudenay, S., Vuagnoux, M.: Password Interception in a SSL/TLS Channel, pp. 583–599. Springer (2003)
15. Chakraborti, A., Mridul, N.: TriviA-ck-v2. https://competitions.cr.yp.to/round2/triviackv2.pdf (2015)
16. Cogliani, S., Maimut, D., Naccache, D., Portella, R., Reyhanitabar, R., Vaudenay, S., Vizar, D.: Offset Merkle-Damgard (OMD) version 2.0. https://competitions.cr.yp.to/round2/omdv20.pdf (2015)
17. Datta, N., Nandi, M.: ELmD v2.0. https://competitions.cr.yp.to/round2/elmdv20.pdf (2015)
18. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: ASCON v1.1. https://competitions.cr.yp.to/round2/asconv11.pdf (2015)
19. Gligoroski, D., Mihajloska, H., Samardjiska, S., Jacobsen, H., El-Hadedy, M., Jensen, R.E., Otte, D.: π-cipher v2.0. https://competitions.cr.yp.to/round2/picipherv20.pdf (2015)
20. Grosso, V., Leurent, G., Standaert, F.X., Varici, K., Journault, A., Durvaux, F., Gaspar, L., Kerckhof, S.: SCREAM: Side-Channel Resistant Authenticated Encryption with Masking. https://competitions.cr.yp.to/round2/screamv3.pdf (2015)
21. Gueron, S.: Intel® advanced encryption standard (aes) new instructions set. Tech. rep., Intel Corporation (2012)
22. Hoang, V.T., Krovetz, T., Rogaway, P.: AEZ v4.1: Authenticated Encryption by Enciphering. https://competitions.cr.yp.to/round2/aezv41.pdf (2015)

23. Homsirikamol, E., Diehl, W., Ferozpuri, A., Farahmand, F., Sharif, M.U., Gaj, K.: GMU Hardware API for Authenticated Ciphers. Cryptology ePrint Archive, Report 2015/669 (2015)
24. Huang, T., Wu, H.: The Authenticated Cipher MORUS (v1). `https://competitions.cr.yp.to/round2/morusv11.pdf` (2015)
25. Iwata, T., Minematsu, K., Guo, J., Morioka, S.: CLOC: Compact Low-Overhead CFB. `https://competitions.cr.yp.to/round2/clocv2.pdf` (2015)
26. Iwata, T., Minematsu, K., Guo, J., Morioka, S., Kobayashi, E.: SILC: SImple Lightweight CFB. `https://competitions.cr.yp.to/round2/silcv2.pdf` (2015)
27. Jean, J., Nikolic, I., Peyrin, T.: Deoxys v1.3. `https://competitions.cr.yp.to/round2/deoxysv13.pdf` (2015)
28. Jean, J., Nikolic, I., Peyrin, T.: Joltik v1.3. `https://competitions.cr.yp.to/round2/joltikv13.pdf` (2015)
29. Krovetz, T.: HS1-SIV (v2). `https://competitions.cr.yp.to/round2/hs1sivv2.pdf` (2015)
30. Krovetz, T., Rogaway, P.: The Software Performance of Authenticated-Encryption Modes, pp. 306–327. Springer (2011)
31. Krovetz, T., Rogaway, P.: OCB (v1). `https://competitions.cr.yp.to/round1/ocbv1.pdf` (2014)
32. Lomont, C.: Introduction to intel® advanced vector extensions. `https://software.intel.com/en-us/articles/introduction-to-intel-advanced-vector-extensions` (2011)
33. Minematsu, K.: AES-OTR v3. `https://competitions.cr.yp.to/round2/aesotrv3.pdf` (2016)
34. Morawiecki, P., Gaj, K., Homsirikamol, E., Matusiewicz, K., Pieprzyk, J., Rogawski, M., Srebrny, M., Wojcik, M.: ICEPOLE v2. `https://competitions.cr.yp.to/round2/icepolev2.pdf` (2015)
35. Nikolic, I.: Tiaoxin-346. `https://competitions.cr.yp.to/round2/tiaoxinv2.pdf` (2015)
36. O. Sarrinen, M.J., Brumley, B.B.: STRIBOBr2: "WHIRLBOB". `https://competitions.cr.yp.to/round2/stribobr2.pdf` (2015)
37. Paoloni, G.: How to benchmark code execution times on intel® ia-32 and ia-64 instruction set architectures. Tech. rep., Intel Corporation (2010)
38. Saarinen, M.J.: The BRUTUS automatic cryptanalytic framework. Journal of Cryptographic Engineering 6(1), 75–82 (2016)
39. Sasaki, Y., Todo, Y., Aoki, K., Naito, Y., Sugawara, T., Murakami, Y., Matsui, M., Hirose, S.: Minalpher v1.1. `https://competitions.cr.yp.to/round2/minalpherv11.pdf` (2015)
40. Siewert, S.: Using Intel® Streaming SIMD Extensions and Intel® Integrated Performance Primitives to Accelerate Algorithms. `https://software.intel.com/en-us/articles/using-intel-streaming-simd-extensions-and-intel-integrated-performance-primitives-to-accelerate-algorithms` (2009)
41. Vaudenay, S.: Security Flaws Induced by CBC Padding — Applications to SSL, IPSEC, WTLS..., pp. 534–545. Springer (2002)
42. Wang, L.: SHELL v2.0. `https://competitions.cr.yp.to/round2/shellv20.pdf` (2015)
43. Wu, H.: ACORN: a Lightweight Authenticated Cipher (vs). `https://competitions.cr.yp.to/round2/acornv2.pdf` (2015)
44. Wu, H., Huang, T.: The JAMBU Lightweight Authentication Encryption Mode (v2). `https://competitions.cr.yp.to/round2/aesjambuv2.pdf` (2015)

45. Wu, H., Preneel, B.: AEGIS: A Fast Authenticated Encryption Algorithm (v1). `https://competitions.cr.yp.to/round1/aegisv1.pdf` (2014)

A Comparison of all Implementations per Candidate

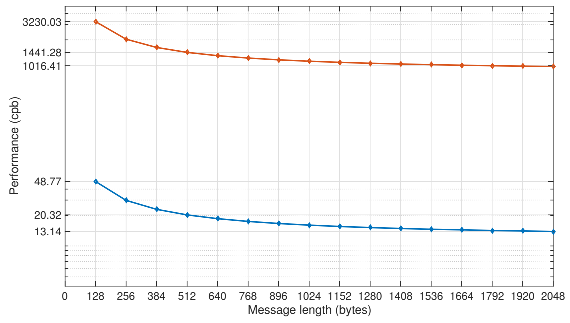


Fig. 4: acorn128v2_opt (blue), acorn128v2_ref (red)

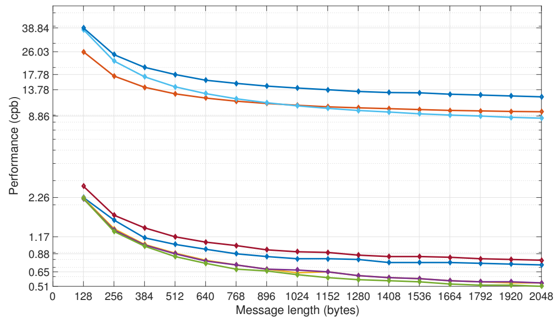


Fig. 5: aegis128_aesni(blue), aegis128_ref (orange), aegis128l_aesnia (yellow), aegis128l_aesnib (purple), aegis128l_aesnic (green), aegis128l_ref (cyan), aegis256_aesni (brown), aegis256_ref (darkblue)

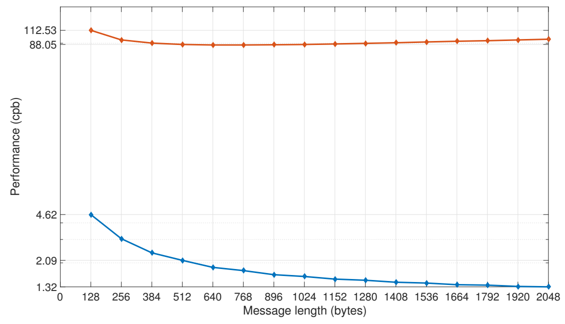


Fig. 6: aezv4_aesni (blue), aezv4_ref (orange)

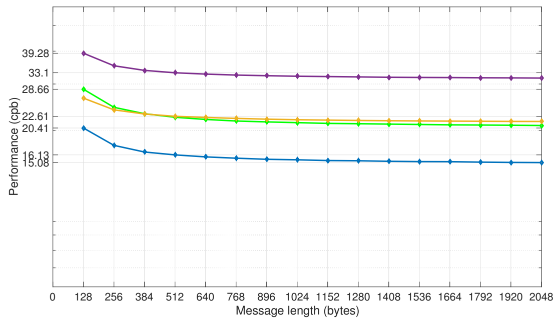


Fig. 7: ascon128av11_opt64 (blue), ascon128av11_ref (green), ascon128v11_opt64 (yellow), ascon128v11_ref (purple)

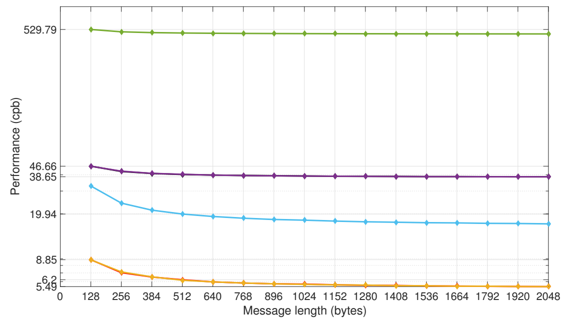


Fig. 8: aes128n12t8clocv2_aesni (red), aes128n12t8clocv2_ref (black), aes128n8t8clocv2_aesni (yellow), aes128n8t8clocv2_ref (purple), twine80n6t4clocv2_ref (green), twine80n6t4clocv2_vperm (blue)

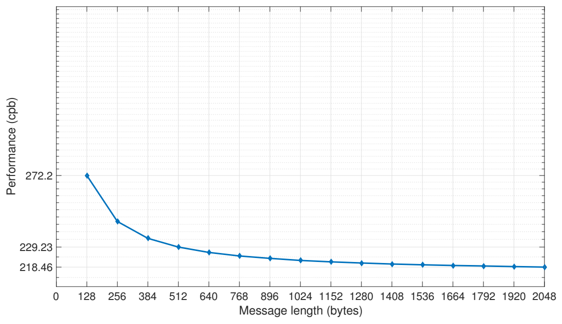


Fig. 9: aescopav2_ref(blue)

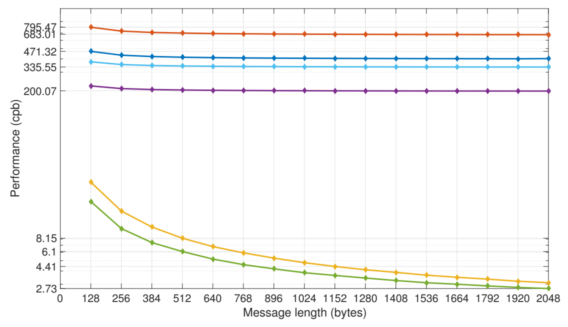


Fig. 10: deoxyseq128128v13_ref (blue), deoxyseq256128v13_ref (orange), deoxysneq128128v13_aesni (yellow), deoxysneq128128v13_ref (purple), deoxysneq256128v13_aesni (green), deoxysneq256128v13_ref (cyan)

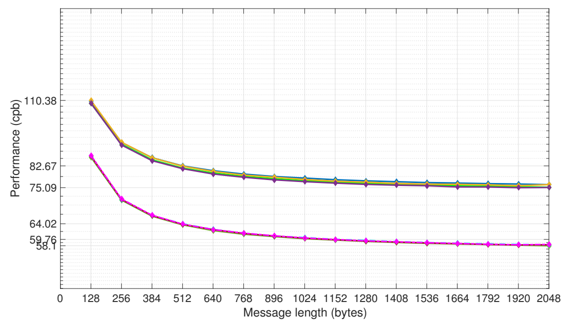


Fig. 11: elmd1000v2_ref (blue), elmd1001v2_ref (green), elmd101270v2_ref(yellow), elmd101271v2_ref (purple), elmd600v2_ref (green), elmd601v2_ref (cyan doted), elmd61279v2_ref (magneta), elmd61271v2_ref (purple dotted)

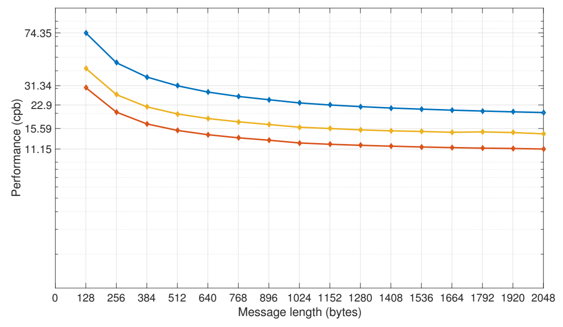


Fig. 12: h1sivhiv1_ref (blue), hs1sivlov1_ref (orange), hs1sivv1_ref (yellow)

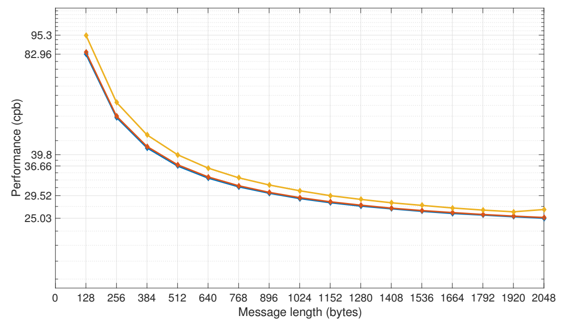


Fig. 13: icepole128av2_ref (blue), icepole128v2_ref (orange), icepole256av2_ref (yellow)

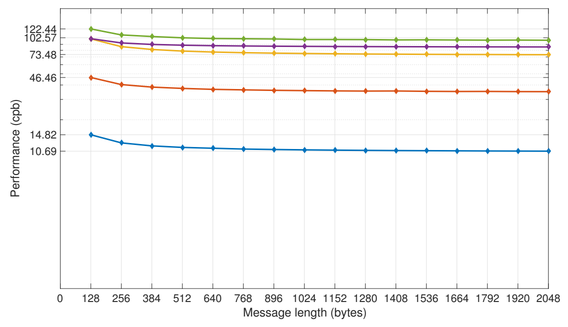


Fig. 14: aesjambuv2_aesni (blue), aesjambuv2_ref (orange), simonjambu128v2_ref (yellow), simonjambu64v2_ref (purple), simonjambu96v2_ref (green)

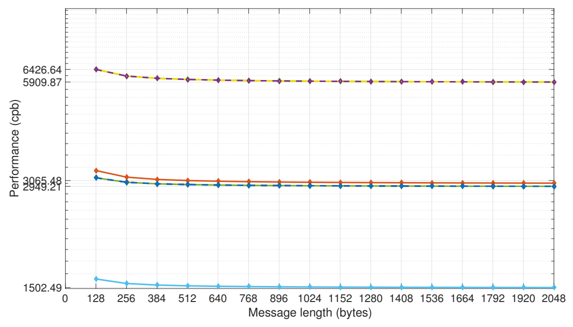


Fig. 15: joltikeq12864v13_ref (light green), joltikeq6464v13_ref (orange), joltikeq80112v13_ref (yellow), joltikeq9696v13_ref (purple), joltikneq12864v13_ref (green), joltikneq6464v13_ref (light blue), joltikneq80112v13_ref (magenta), joltikneq9696v13_ref (blue)

Fig. 16: ketjejrv1_compact (blue), ketjejrv1_reference (orange), ketjesrv1_compact (yellow), ketjesrv1_reference (purple)

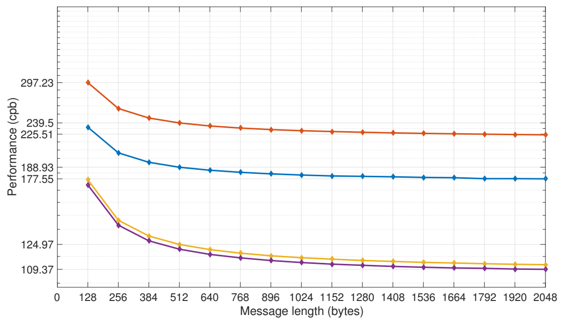


Fig. 17: lakekeyakv2_compact (blue), lakekeyakv2_generic32 (red dotted), lakekeyakv2_generic32lc (cyan dotted), lakekayakv2_generic64 (purple), lakekeyakv2_generic64lc (green dotted)

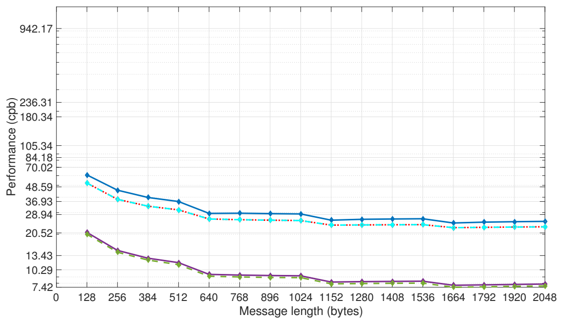


Fig. 18: riverkeyakv2_compact (orange), riverkeyakv2_generic32 (yellow dotted), riverkeyak_generic32lc (purple), riverkeyakv2_generic64 (green), riverkeyakv2_generic64lc (red), riverkeyakv2_reference (brown), riverkeyakv2_reference32bits (blue dotted)

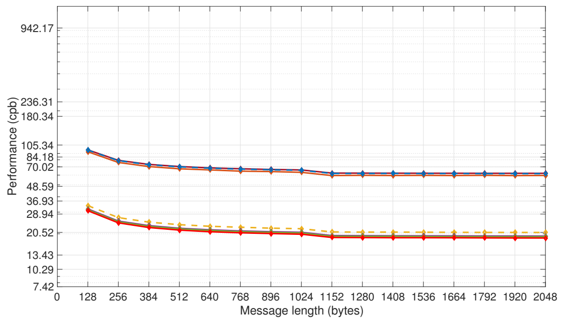


Fig. 19: seakeyakv2_compact (orange), seakeyakv2_generic32 (yellow dotted), seakeyakv2_generic32lc (purple dotted), seakeyakv2_generic64 (dark blue), seakeyakv2_generic64lc (cyan dotted)

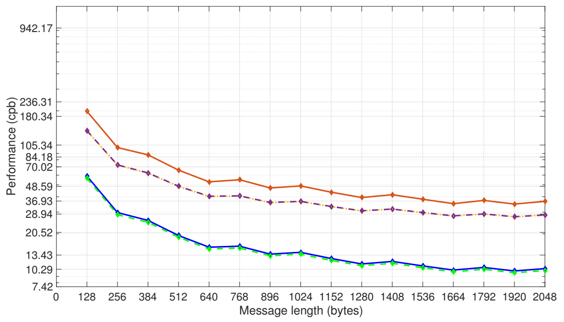


Fig. 20: lunarkeyakv2_compact (light blue), lunarkeyakv2_generic32 (magenta dotted), lunarkeyakv2_generic32lc (blue dotted), lunarkeyav2_generic64 (orange), lunarkeyakv2_generic64lc (pink)

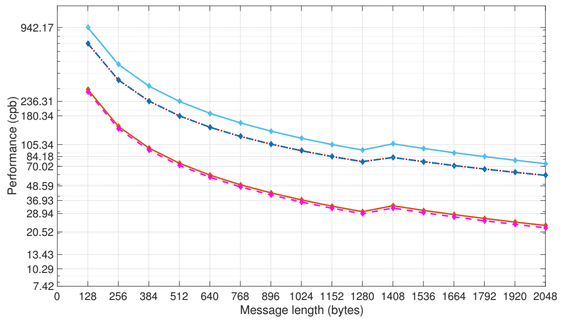


Fig. 21: oceankeyakv2_compact (purple), oceankeyakv2_generic32 (green dotted), oceankeyakv2_generic32lc (light blue dotted), oceankeyakv2_generic64 (magenta dotted), oceankeyakv2_generic64lc (blue)

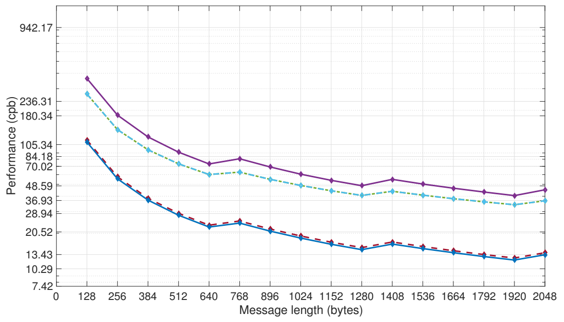


Fig. 22: minalpherv11_ref (blue)

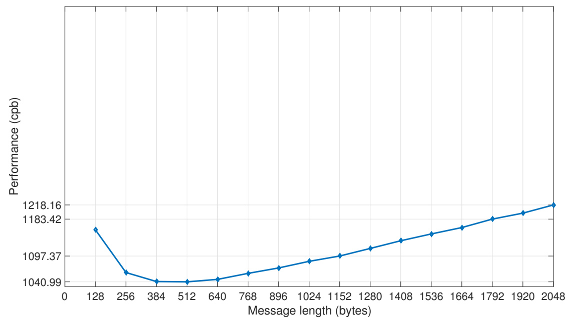


Fig. 23: morus1280128v1_ref (blue), morus1280128v1_ref64 (orange), morus1280128v1_sse2 (yellow), morus1280256v1_ref (purple), morus1280256v1_ref64 (green), morus1280256v1_sse2 (cyan), morus6400128v1_sse2 (brown)

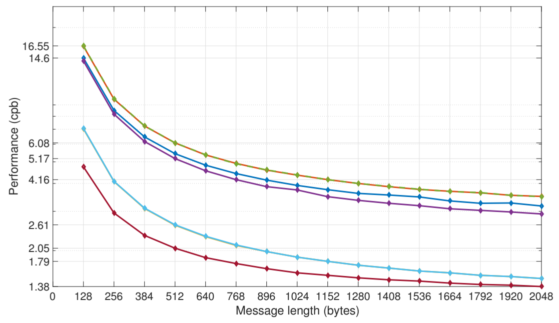


Fig. 24: norx1641_ref (blue), norx3241_ref (light green), norx3241_xmm (olive), norx3261_ref (purple), norx3261_xmm (green), norx6441_ref (light blue), norx6441_xmm (magenta), norx6444_ref (gray), norx6461_ref (orange), norx6461_xmm (yellow)

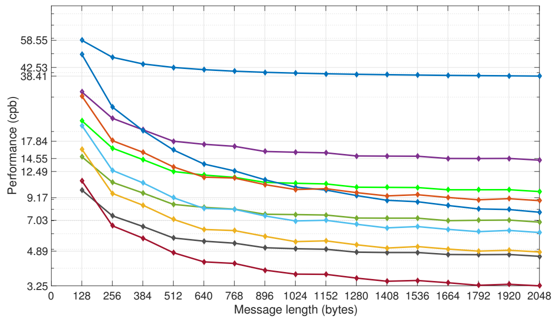


Fig. 25: aes128ocbtaglen128v1_opt (blue), aes128ocbtaglen128v1_ref (green), aes128ocbtaglen64v1_ref (orange), aes128ocbtaglen96v1_ref (purple), aes192ocbtaglen128v1_opt (olive), aes192ocbtaglen128v1_ref (cyan), aes192ocbtaglen64v1_ref (purple dotted), aes192ocbtaglen96v1_ref (pink), aes256ocbtaglen128v1_opt (brown), aes256ocbtaglen128v1_ref (yellow), aes256ocbtaglen64v1_ref (violett), aes256ocbtaglen96v1_ref (green dotted)

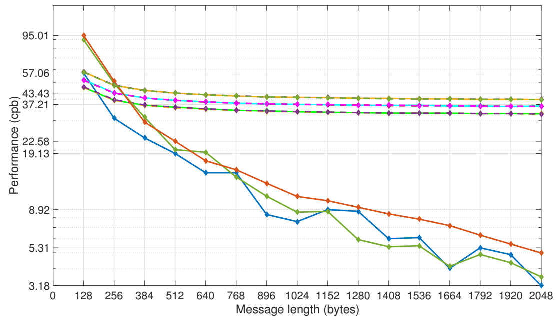


Fig. 26: aes128otrpv2_ref (blue), aes128otrsv2_ref (orange), aes256otrpv2_ref (yellow), aes256otrsv2_ref (purple)

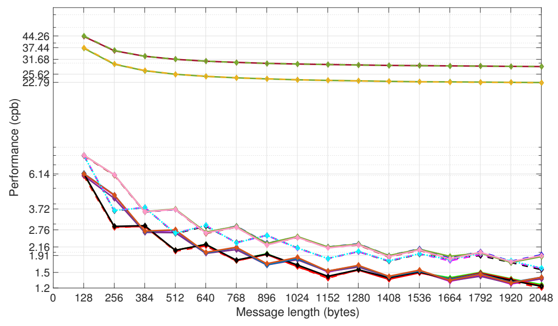


Fig. 27: paeq128_aesni (blue dotted), paeq128_ref (dark blue), paeq128t_aesni (orange), paeq128t_ref (green), paeq128tnm_aesni(olive dotted), paeq128tnm_ref (cyan dotted), paeq160_aesni (magenta), paeq160_ref (blue), paeq64_aesni (blue), paeq64_ref (orange), paeq80_aesni (purple), paeq80_ref (yellow)

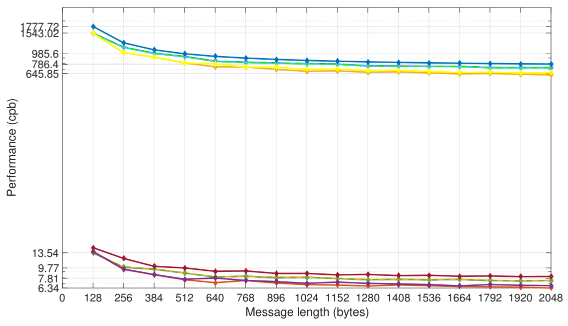


Fig. 28: omdsha256k128n96tau128v2_avx1 (black), omdsha256k128n96tau128v2_ref (pink dotted), omdsha256k128n96tau128v2_sse4 (orange dotted), omdsha256k128n96tau64v2_avx1 (gray), omdsha256k128n96tau64v2_ref (green), omdsha256k128n96tau64v2_sse4 (pink), omdsha256k128n96tau96v2_avx1 (brown), omdsha256k128n96tau96v2_ref (blue), omdsha256k128n96tau96v2_sse4 (red), omdsha256k192n104tau128v2_avx1 (orange), omdsha256k192n104tau128v2_ref (purple dotted), omdsha256k192n104tau128v2_sse4 (olive), omdsha256k256n104tau160v2_avx1 (light blue), omdsha256k256n104tau160v2_ref (brown), omdsha256k256n104tau160v2_sse4 (dark blue), omdsha256k256n248tau256v2_avx1 (light brown), omdsha256k256n248tau256v2_ref (yellow dotted), omdsha256k256n248tau256v2_sse4 (purple dotted)

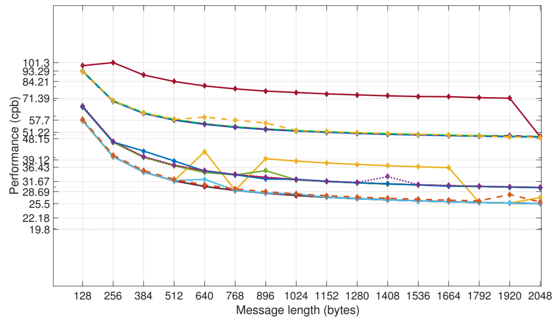


Fig. 29: omdsha512k128n128tau128v2_avx1 (violett), omdsha512k128n128tau128v2_ref (light blue), omdsha512k128n128tau128v2_sse4 (purple dotted), omdsha512k256n256tau256v2_avx1 (blue), omdsha512k256n256tau256v2_ref (light brown), omdsha512k256n256tau256v2_sse4 (black dotted), omdsha512k512n256tau256v2_avx1 (green dotted), omdsha512k512n256tau256v2_ref (cyan dotted), omdsha512k512n256tau256v2_sse4 (light blue dotted)

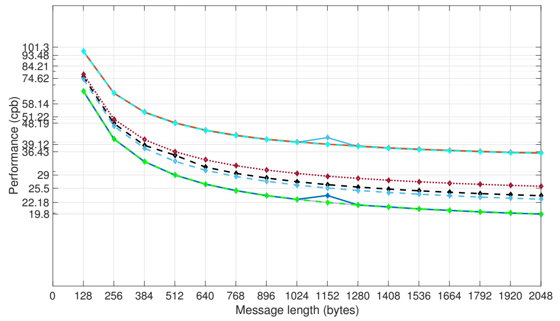


Fig. 30: pi16cipher096v2_ref (blue), pi16cipher128v2_ref (green), pi32cipher128v2_ref (yellow) pi32cipher256v2_ref (purple), pi64cipher128v2_ref (olive), pi64cipher256v2_ref (light blue)

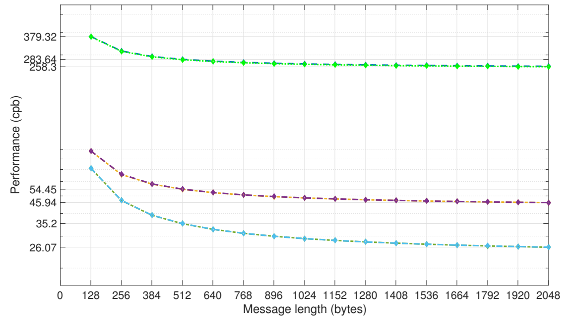


Fig. 31: aes128poetv2aes128ls0lt0_ni (blue), aes128poetv2aes128ls0lt0_ref (orange), aes128poetv2aes128ls128lt128_ref (yellow), aes128poetv2aes4ls0lt0_ni (purple), aes128poetv2aes4ls0lt0_ref (green), aes128poetv2aes4ls128lt128_ref (cyan)

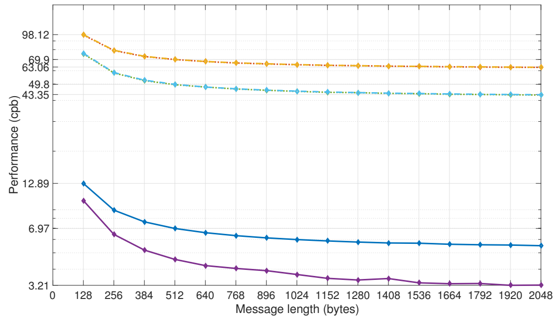


Fig. 32: primatesv1ape120_ref (blue), primatesv1ape80_ref (orange), primatesv1gibbon120_ref (yellow), primatesv1gibbon80_ref (purple), primatesv1hanuman120_ref (green), primatesv1hanuman80_ref (cyan)

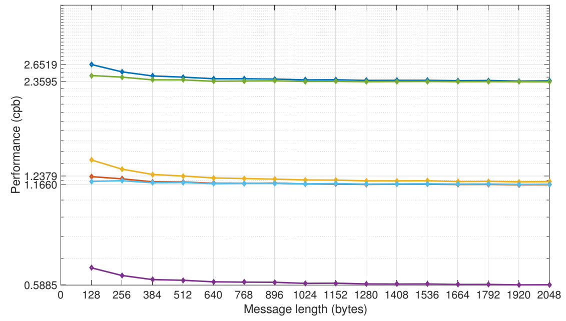


Fig. 33: scream10v3_ref (blue), scream10v3_sse (orange), scream12v3_ref (yellow), scream12v3_ref (purple)

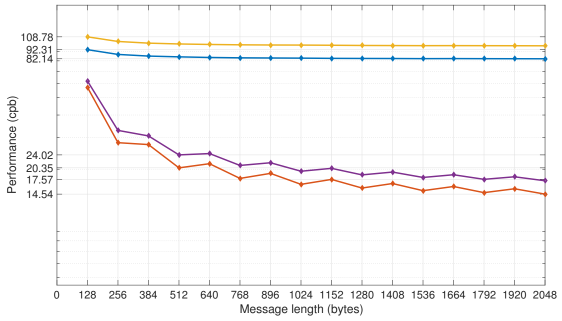


Fig. 34: shellaes128v2d4n64_ref (blue dotted), shellaes128v2d4n80_ref (orange), shellaes128v2d5n64_ref (yellow dotted), shellaes128v2d5n80_ref (purple), shellaes128v2d6n64_ref (green dotted), shellaes128v2d6n80_ref (cyan), shellaes128v2d7n64_ref (light green), shellaes128v2d7n80_ref (blue), shellaes128v2d8n64_ref (orange dotted), shellaes128v2d8n80_ref (yellow)

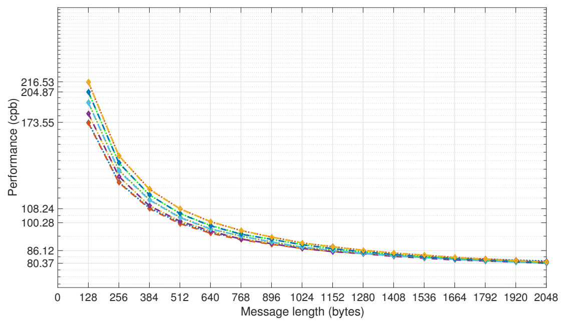


Fig. 35: aes128n12t8silcv2_aesni (blue), aes128n12t8silcv2_ref (green), aes128n8t8silcv2_aesni (yellow), aes128n8t8silcv2_ref (purple)

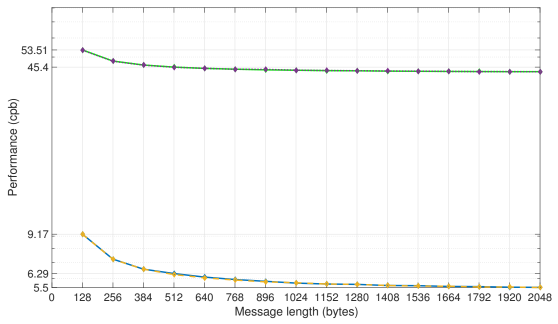


Fig. 36: stribob192r2_8bit (blue), stribob192r2_bitslice (orange), stribob192r2_ref (yellow), stribob192r2_smaller (purple), stribob192r2_ssse3 (green)

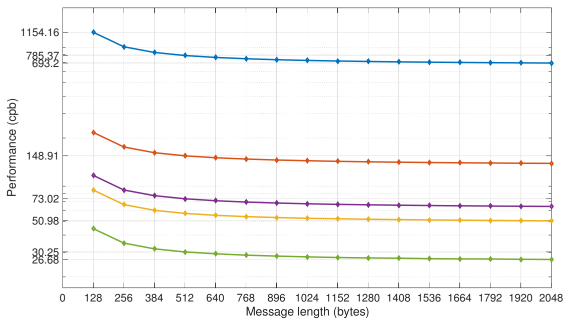


Fig. 37: tiaoxinv2_nim (blue), tiaoxinv2_ref (orange)

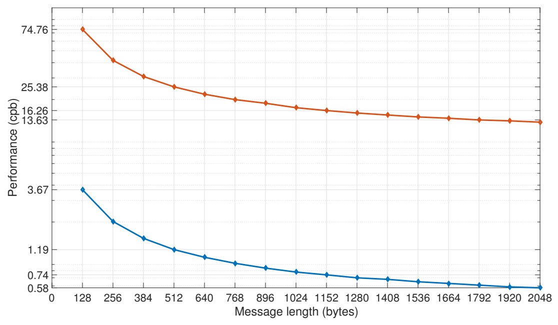
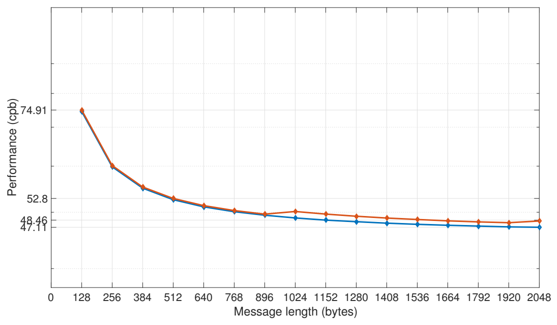


Fig. 38: trivia0v2_ref (blue), trivia128v2_ref (orange)



B 3D Plots for Increasing Message and Associated Data Lengths

Fig. 39: acorn128v2_opt

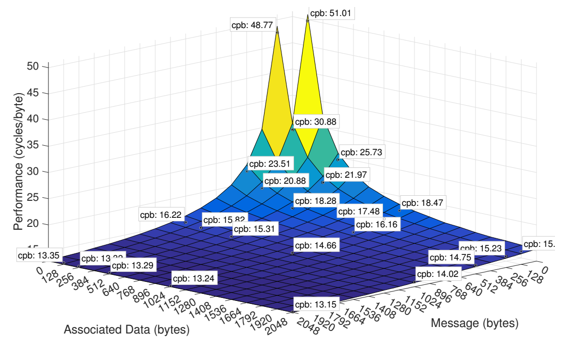


Fig. 40: aeadaes256ocbtaglen128v1_opt

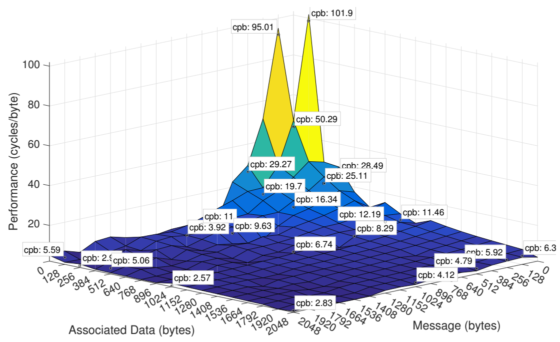


Fig. 41: aegis128l_aesnic

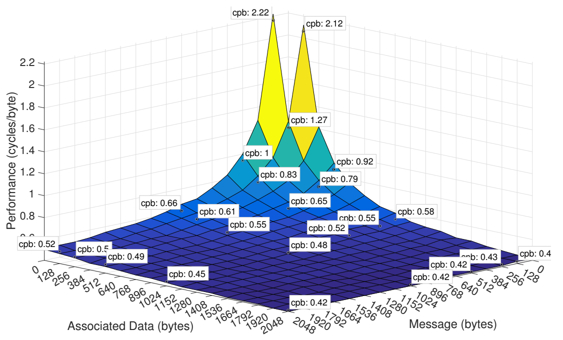


Fig. 42: aes128n8t8clocv2_aesni

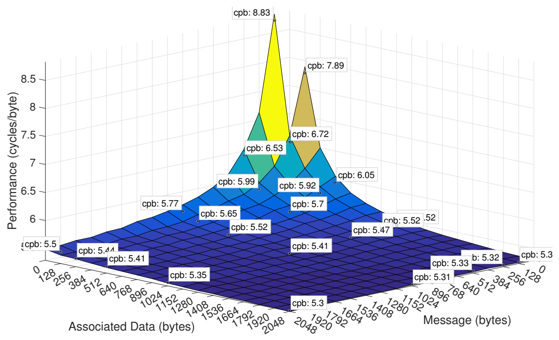


Fig. 43: aes128n8t8silcv2_aesni

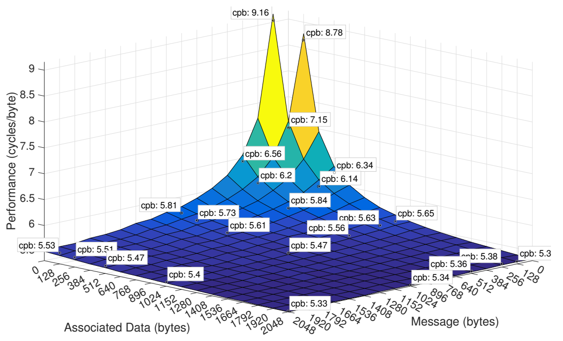


Fig. 44: aescopav2_ref

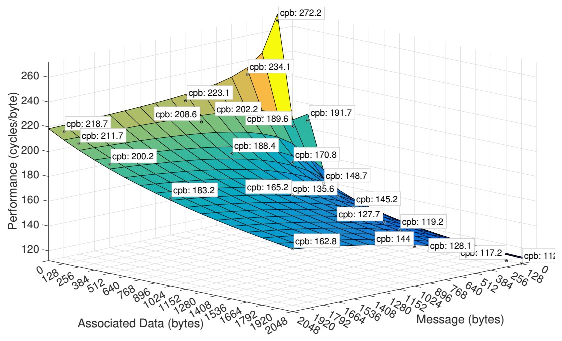


Fig. 45: aesjambuv2_aesni

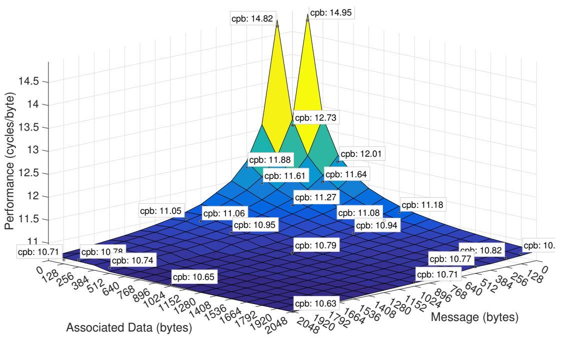
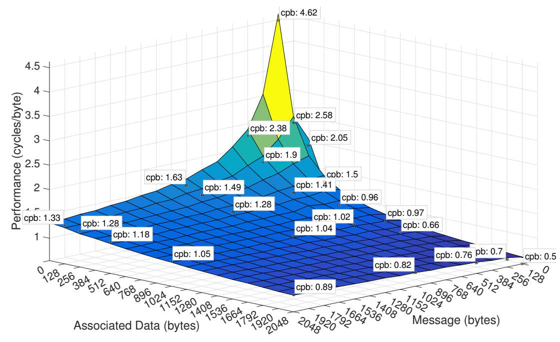


Fig. 46: aezv4_aesni



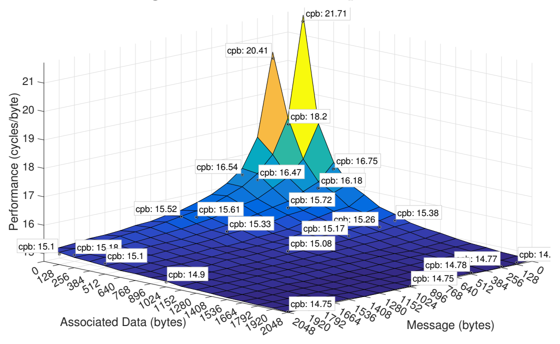


Fig. 47: ascon128av11_opt64

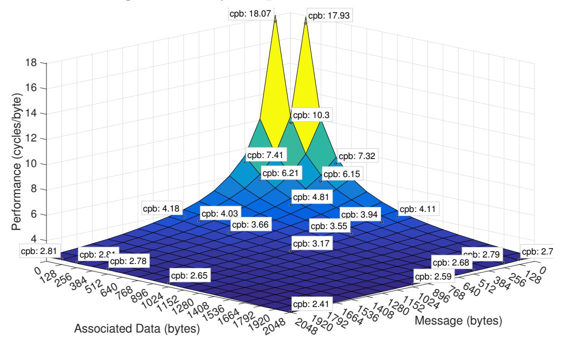


Fig. 48: deoxysneq256128v13_aesni

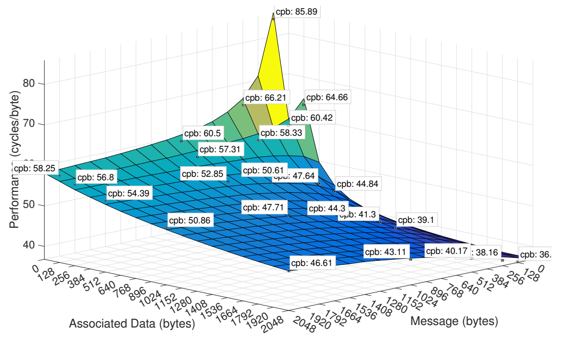


Fig. 49: elmd600v2_ref

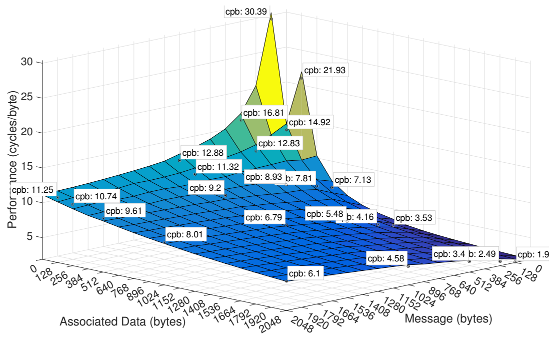


Fig. 50: hs1sivlov1_ref

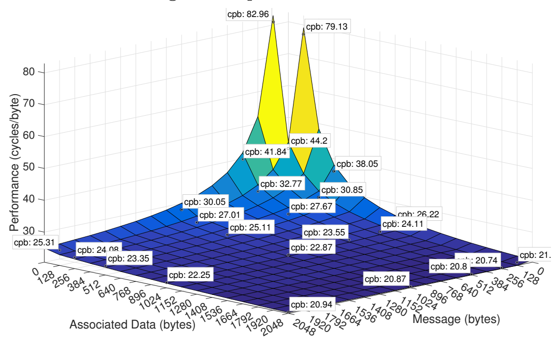


Fig. 51: icepole128av2_ref

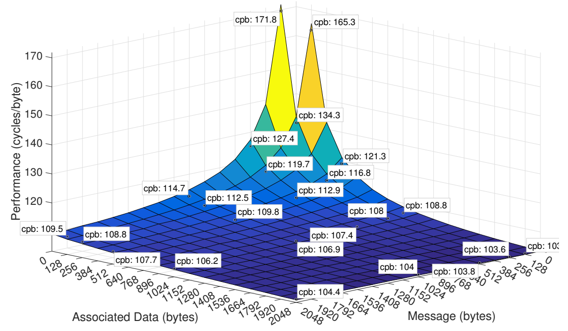


Fig. 52: ketjesrv1_reference

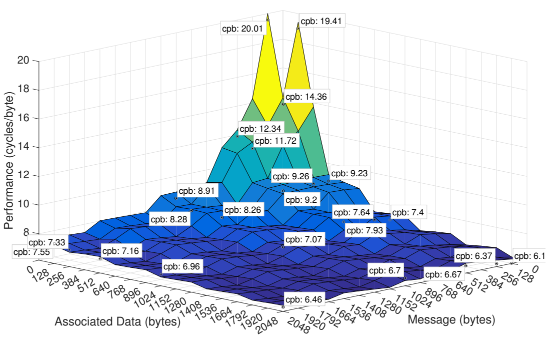


Fig. 53: lakekeyakv2_generic64lc

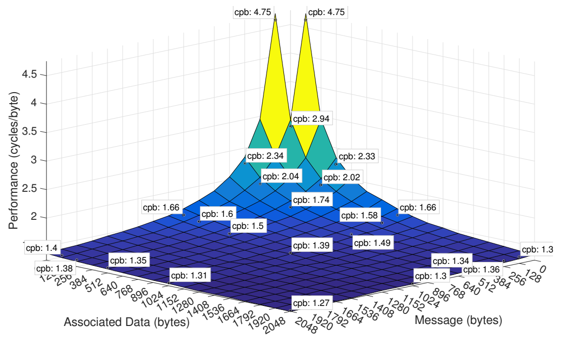


Fig. 54: morus640128v1_sse2

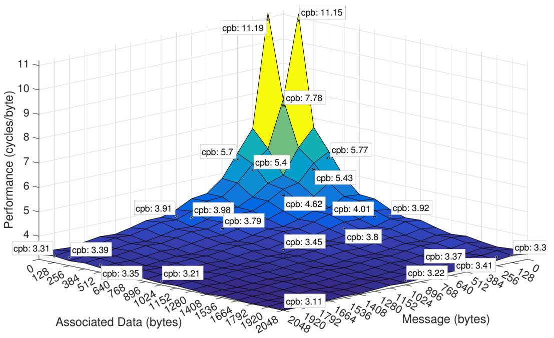


Fig. 55: norx6441_xmm

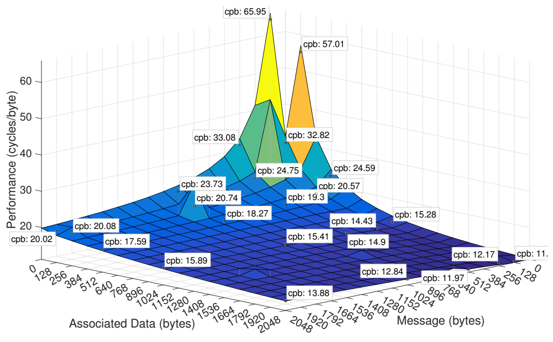


Fig. 56: omdsha512k512n256tau256v2_avx1

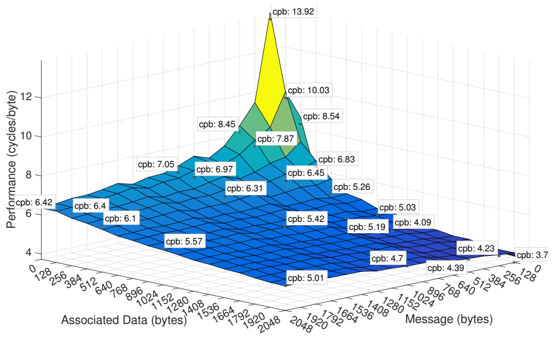


Fig. 57: paeq64_aesni

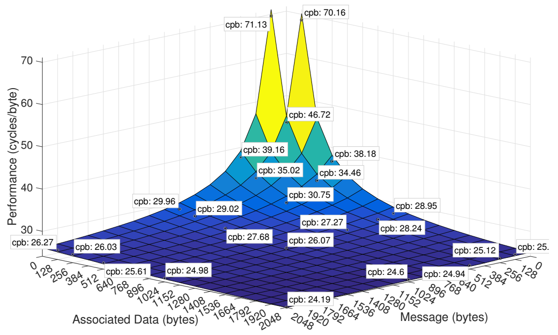


Fig. 58: pi64cipher128v2_ref

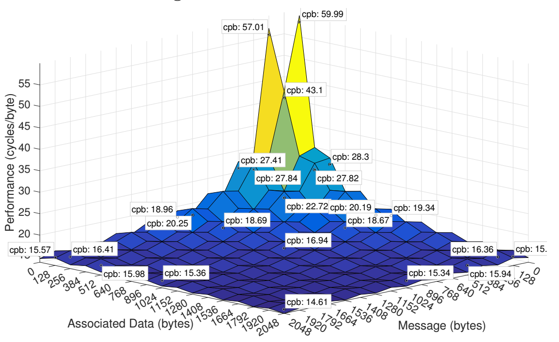


Fig. 59: scream10v3_sse

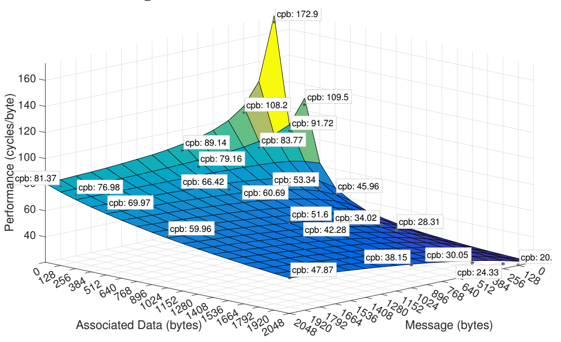


Fig. 60: shellaes128v2d4n80_ref

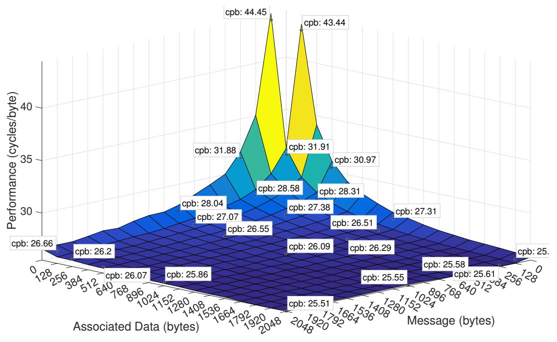


Fig. 61: stribob192r2_ssse3

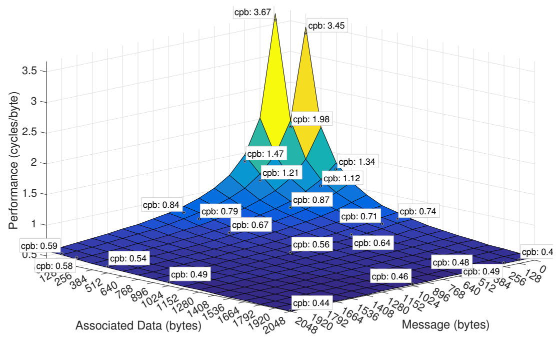


Fig. 62: tiaoxinv2_nim

Fig. 63: trivia0v2_ref

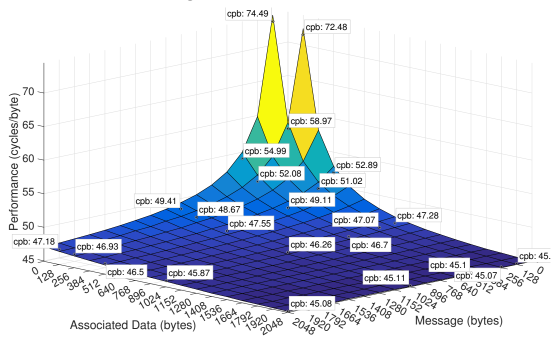


Fig. 64: aes128otrsv2_ref

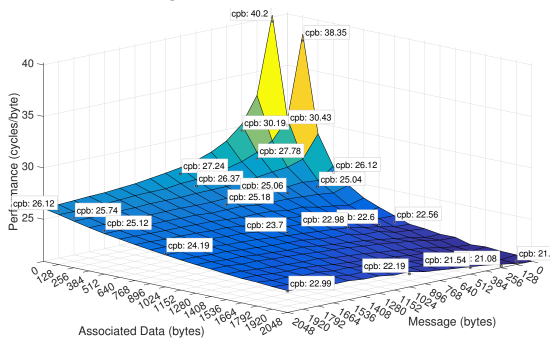
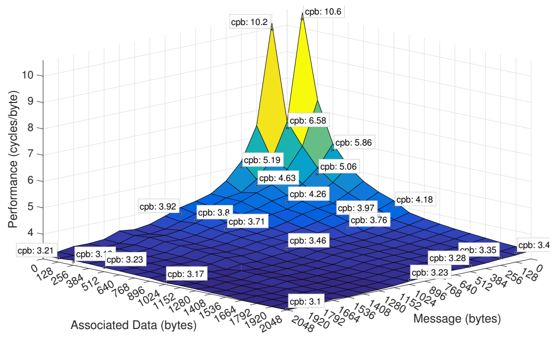


Fig. 65: aes128poetv2aes4ls0lt0_ni



C Bar Charts for Real-World Message Sizes for each CAESAR Candidate

Fig. 66: acorn128v2_opt

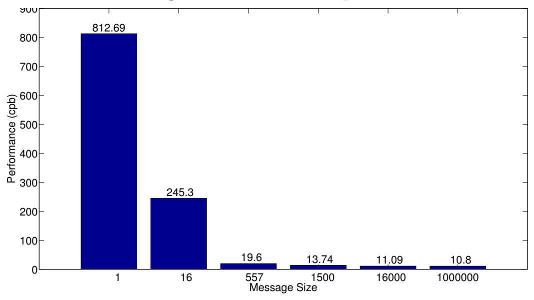


Fig. 67: aeadaes256ocbtaglen128v1_opt

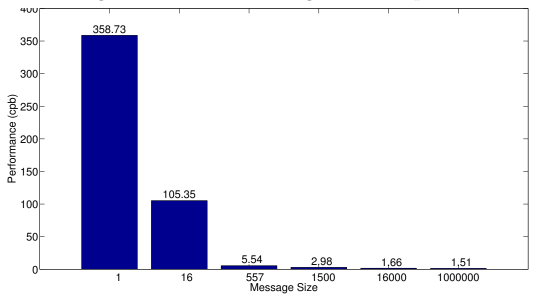


Fig. 68: aegis128l_aesnic

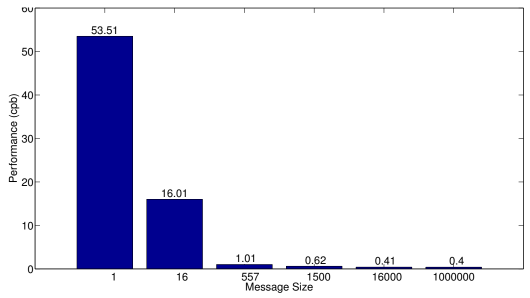


Fig. 69: aes128n8t8clocv2_aesni

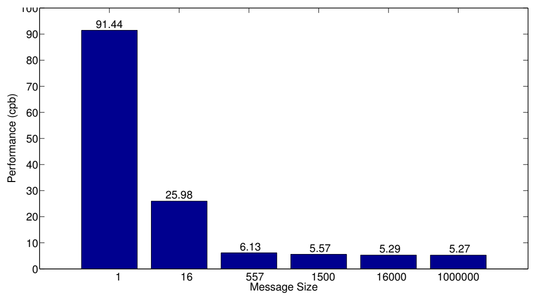


Fig. 70: aes128n8t8silcv2_aesni

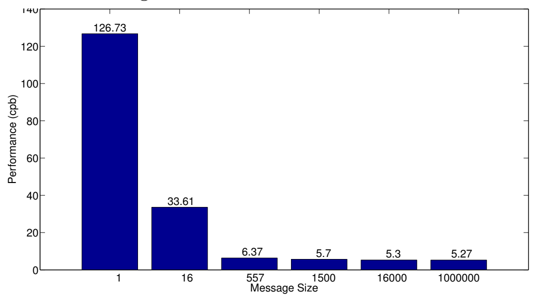


Fig. 71: aes128otrsv2_ref

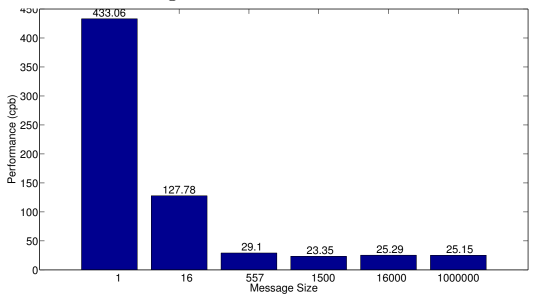


Fig. 72: aes128poetv2aes4ls0lt0_ni

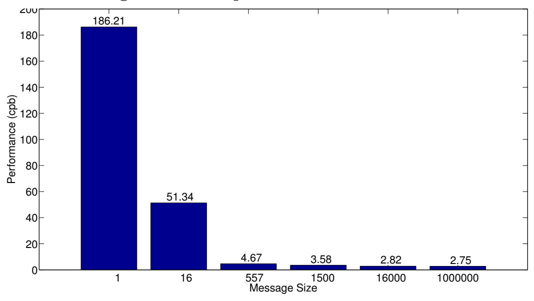


Fig. 73: aescopav2_ref

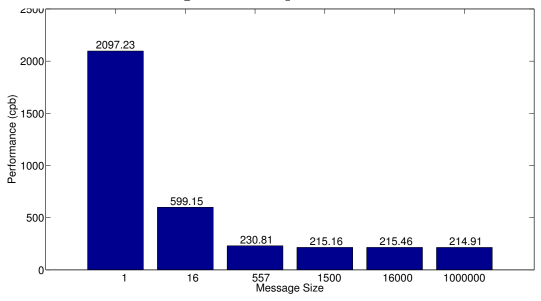


Fig. 74: aesjambuv2_aesni

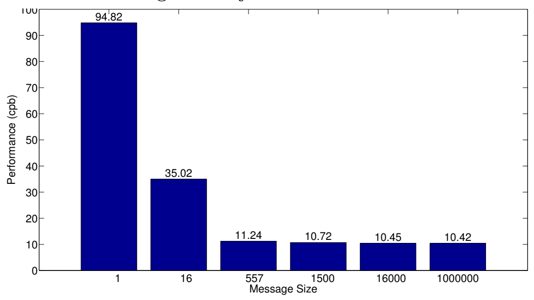


Fig. 75: aezv4_aesni

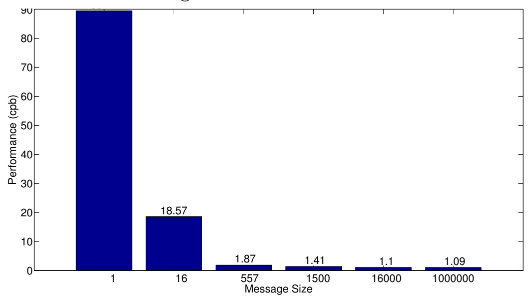


Fig. 76: ascon128av11_opt64

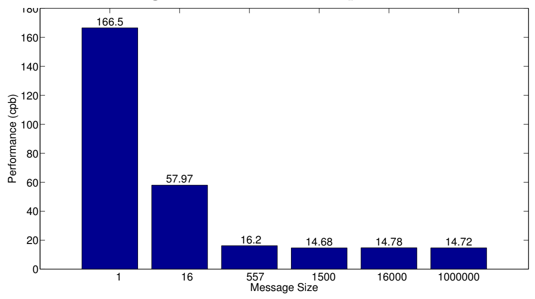


Fig. 77: deoxysneq256128v13_aesni

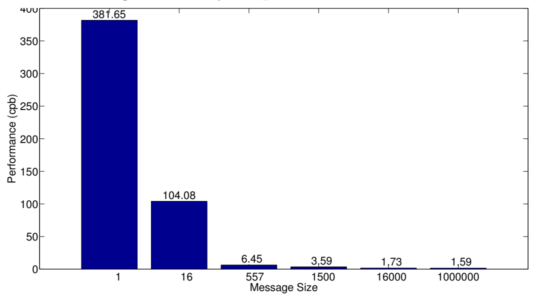


Fig. 78: elmd600v2_ref

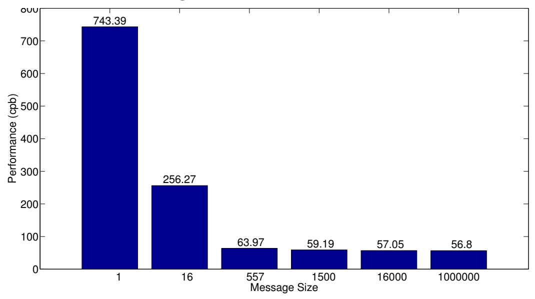


Fig. 79: hs1sivlov1_ref

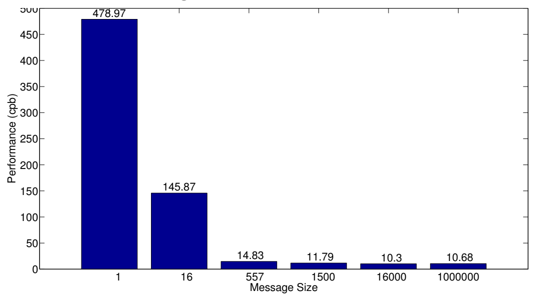


Fig. 80: icepole128av2_ref

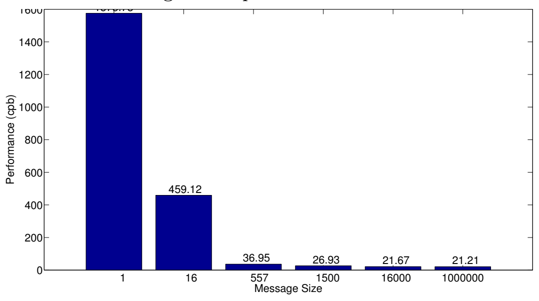


Fig. 81: ketjesrv1_reference

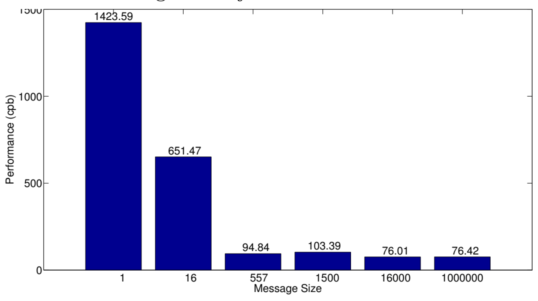


Fig. 82: lakekeyakv2_generic64lc

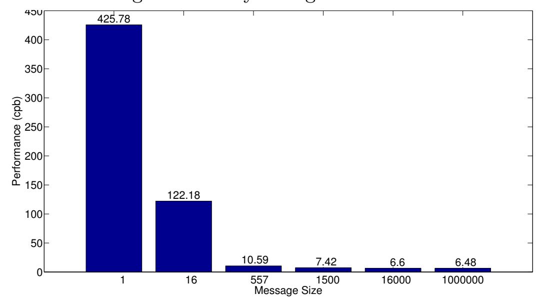


Fig. 83: morus640128v1_sse2

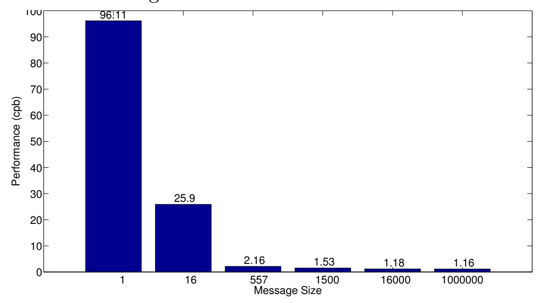


Fig. 84: norx6441_xmm

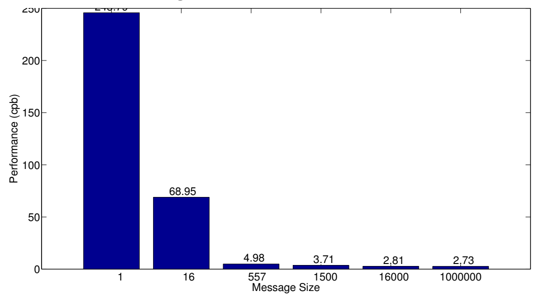


Fig. 85: omdsha512k512n256tau256v2_avx1

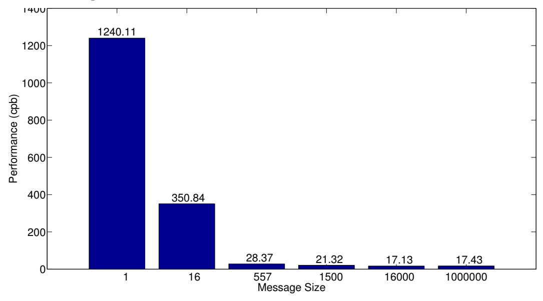


Fig. 86: paeq64_aesni

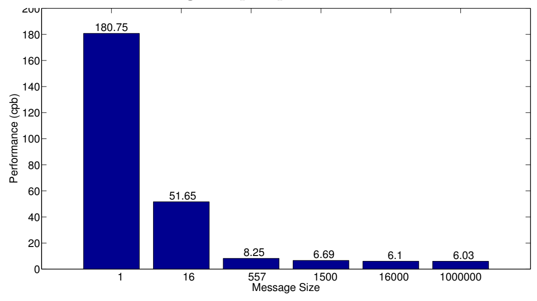


Fig. 87: pi64cipher128v2_ref

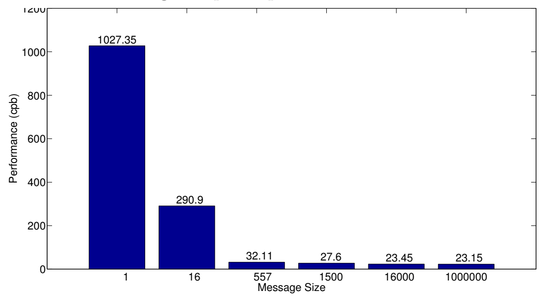


Fig. 88: scream10v3_sse

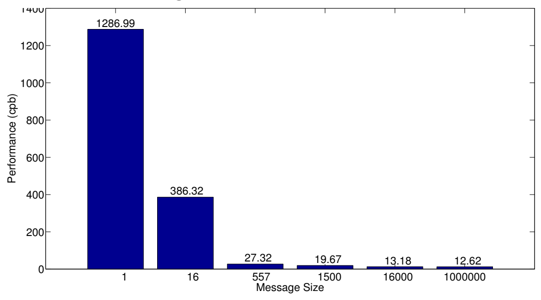


Fig. 89: shellaes128v2d4n80_ref

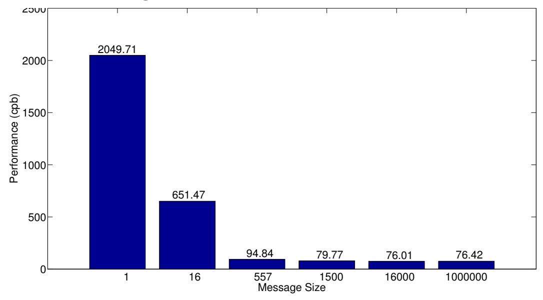


Fig. 90: stribob192r2_ssse3

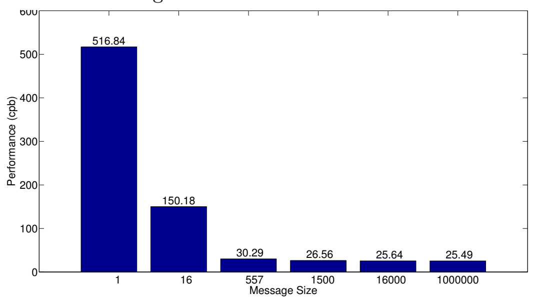


Fig. 91: tiaoxinv2_nim

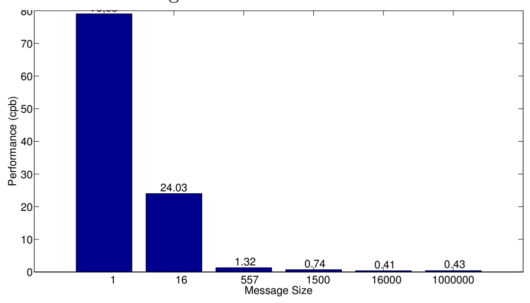
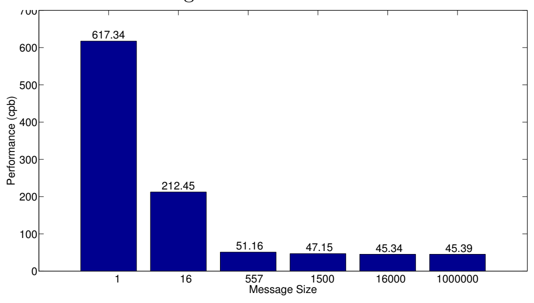


Fig. 92: trivia0v2_ref


Performance (cpb)
Message Size