Adding more parallelism to the AEGIS authenticated encryption algorithms

Frank Denis

Fastly Inc.
fde@00f.net

Abstract. While the round function of the AEGIS authenticated encryption algorithms is highly parallelizable, their mode of operation is not.

We introduce two new modes to overcome that limitation: AEGIS-128X and AEGIS-256X, that require minimal changes to existing implementations and retain the security properties of AEGIS-128L and AEGIS-256.

1 Introduction

AEGIS [WP14] is a family of three authenticated encryption algorithms, originally designed to leverage the AES-NI instructions set introduced by Intel in 2010 [ADF+10]. These instructions perform several compute intensive parts of the AES algorithm, significantly improving the performance of software AES implementations while minimizing the risks of side channel attacks.

However, concurrent AES round instructions are required to fully utilize the AES pipelines. The AEGIS round function was specifically designed with this in mind, and allows up to 8 AES blocks to be updated concurrently. Its design made it the fastest candidate of the CAESAR competition on Intel CPUs with hardware AES acceleration [ARAR16].

Nonetheless, the mode of operation is similar to a duplex: after its initialization, the state is recursively updated. That effectively limits the parallelism of the construction to the parallelism of the round function.

In [BLT15], Bogdanov, Lauridsen, and Tischhauser made a similar observation regarding multiple candidates of the CAESAR competition. They proposed a novel "comb scheduler" to process multiple messages simultaneously.

The modes presented here also encrypt multiple messages simultaneously using the same cipher, but assume that they are fragments of the same message, and share the same key, initialization vector and length.

Given a parallelism degree ν, an input message is split into ν evenly distributed parts, that can be encrypted concurrently. The resulting ciphertexts are then combined, as well as their authentication tags.

The underlying encryption algorithms remain the existing AEGIS algorithms, with a minor addition to their initialization functions.

2 Operations, Variables and Functions

The operations, variables and functions used in this document are defined below.

2.1 Operations

$\|x\|$	: Size of x in bits
$x \oplus y$	: Bit-wise exclusive OR
$\mathbb{F}_{128L}(S, m_0, m_1)$	: AEGIS-128L state update function
$\mathbb{F}_{256}(S, m)$	: AEGIS-256 state update function
$x \| y$	: Concatenation of x and y
$Pad(x, \ell)$	: Add trailing 0 bits to pad x to ℓ bits
$Enc_{128L}(CTX, K, IV, A, M)$	: AEGIS-128L encryption function with context separation
$Enc_{256}(CTX, K, IV, A, M)$	: AEGIS-256 encryption function with context separation
$Enc_{128X}[\nu](K, IV, A, M)$	: AEGIS-128X[ν] parallel encryption function
$Enc_{256X}[\nu](K, IV, A, M)$	: AEGIS-256X[ν] parallel encryption function
$Trunc(x, \ell)$	: Truncate x to the first ℓ bits

2.2 Variables and constants

AD	: Associated data or {} if unspecified
AD_i	: 128-bit associated data block
$\hat{AD}$	: AD, padded to $r \cdot \nu$ bits
$\hat{AD}_i$	: 128-bit associated data block
$\overline{AD}_i$	: $\hat{AD}$ fragment ($\frac{\|\hat{AD}\|}{\nu}$ bits)
$\overline{AD}_{i,j}$	: 128-bit $\hat{AD}$ fragment block
C	: Ciphertext
$\hat{C}$	: C, padded to $r \cdot \nu$ bits
$\hat{C}_i$	: A 128-bit ciphertext block
$\overline{C}_i$	: C fragment ($\frac{\|\hat{C}\|}{\nu}$ bits)
$\overline{C}_{i,j}$	: 128-bit C fragment block
$const_0$	: First half of the AEGIS constant (128 bits)
$const_1$	: Second half of the AEGIS constant (128 bits)
CTX	: Context separator
K_{128}	: 128-bit encryption key (AEGIS-128, -128L)
K_{256}	: 256-bit encryption key (AEGIS-256)
$K_{256,0}$	: First half of a 256-bit key
$K_{256,1}$	: Second half of a 256-bit key
P	: Plaintext
$\hat{P}$	: P, padded to $r \cdot \nu$ bits
$\hat{P}_i$	: 128-bit plaintext block
$\overline{P}$	: P fragment ($\frac{\|\hat{P}\|}{\nu}$ bits)
$\overline{P}_i$	: 128-bit P fragment block
IV_{128}	: 128-bit initialization vector (AEGIS-128, -128L)
IV_{256}	: 256-bit initialization vector (AEGIS-256)
$IV_{256,0}$	: First half of a 256-bit initialization vector
$IV_{256,1}$	: Second half of a 256-bit initialization vector
ν	: Parallelism degree (≥ 1)
r	: Absorption rate (128 or 256 bits)
S	: AEGIS state
S_i	: A 128-bit AEGIS state block
T	: Authentication tag for C
$\overline{T}_i$	: Authentication tag for $\overline{C}_i$

3 Context separation

From an application perspective, new AEGIS variants should ideally share the same interface as existing variants. Namely, they should accept a single message, optional associated data, a 128 or 256 bit key, and a 128 or 256 bit initialization vector.

However, AEGIS is meant to be used in a nonce-respecting scenario [VV18]. Clearly, reusing the same key and IV to encrypt different parts of a message would violate that contract.

In order to avoid universal forgery and decryption attacks, we augment the AEGIS initialization functions with a context, meant to provide domain separation. That is, for two different values $CTX_0 \neq CTX_1$, $Enc.[\cdot](CTX_0, \cdot)$ and $Enc.[\cdot](CTX_1, \cdot)$ act as two different functions of $\{K, IV, A, M\}$.

3.1 Augmenting AEGIS-128L for context separation

AEGIS-128L defines the initial state S as a vector of eight AES blocks $\{S_0, S_1, \dots S_7\}$ set to:

Block	Initial value
S_0	$K_{128} \oplus IV_{128}$
S_1	$const_1$
S_2	$const_0$
S_3	$const_1$
S_4	$K_{128} \oplus IV_{128}$
S_5	$K_{128} \oplus const_0$
S_6	$K_{128} \oplus const_1$
S_7	$K_{128} \oplus const_0$

From this state, the original AEGIS-128L initialization function performs 10 updates as described in algorithm 1.

Algorithm 1 Contextless AEGIS-128L initialization

function INITIALIZE(K, IV)
  $S \leftarrow \{K_{128} \oplus IV_{128}, const_1, const_0, const_1, K_{128} \oplus IV_{128}, K_{128} \oplus const_0, K_{128} \oplus const_1, K_{128} \oplus const_0\}$
  $i \leftarrow 0$
  while $i < 10$ **do**
    $S \leftarrow \mathbb{F}_{128L}(S, IV_{128}, K_{128})$
    $i \leftarrow i + 1$
  end while
end function

We augment this function to accept a context parameter CTX. Before each update, the context is added to the blocks at indices 3 and 7 of the state, as described in algorithm 2

Note that when $CTX = 0$, the resulting state is exactly the same as AEGIS-128L, as originally specified, without a context.

Algorithm 2 AEGIS-128L initialization with context

function INITIALIZE(CTX, K, IV)
  $S \leftarrow \{K_{128} \oplus IV_{128}, const_1, const_0, const_1, K_{128} \oplus IV_{128}, K_{128} \oplus const_0, K_{128} \oplus const_1, K_{128} \oplus const_0\}$
  $i \leftarrow 0$
  while $i < 10$ **do**
    $S_3 \leftarrow S_3 \oplus Pad(CTX, 128)$
    $S_7 \leftarrow S_7 \oplus Pad(CTX, 128)$
    $S \leftarrow \mathbb{F}_{128L}(S, IV_{128}, K_{128})$
    $i \leftarrow i + 1$
  end while
end function

3.2 Augmenting AEGIS-256 for context separation

AEGIS-256 accepts a 256-bit key K_{256} made of two AES blocks $\{K_{256,0}, K_{256,1}\}$, as well as 256-bit initialization vector IV_{256} made of two AES blocks $\{IV_{256,0}, IV_{256,1}\}$.

The initial state S is a vector of six AES blocks $\{S_0, S_1, \dots S_5\}$ set to:

Block	Initial value
S_0	$K_{256,0} \oplus IV_{256,0}$
S_1	$K_{256,1} \oplus IV_{256,1}$
S_2	$const_0$
S_3	$const_1$
S_4	$K_{256,0} \oplus const_1$
S_5	$K_{256,1}1 \oplus const_0$

From this state, the original AEGIS-256 initialization function performs 16 updates as described in algorithm 3.

Algorithm 3 Contextless AEGIS-256 initialization

function INITIALIZE(K, IV)
  $S \leftarrow \{K_{256,0} \oplus IV_{256,0}, K_{256,1} \oplus IV_{256,1}, const_0, const_1, K_{256,0} \oplus const_1, K_{256,1} \oplus const_0\}$
  $i \leftarrow 0$
  while $i < 4$ **do**
    $S \leftarrow \mathbb{F}_{256}(S, K_{256,0})$
    $S \leftarrow \mathbb{F}_{256}(S, K_{256,1})$
    $S \leftarrow \mathbb{F}_{256}(S, IV_{256,0})$
    $S \leftarrow \mathbb{F}_{256}(S, IV_{256,1})$
    $i \leftarrow i + 1$
  end while
end function

We augment this function to accept a context parameter CTX. Before each update, the context is added to the blocks at indices 3 and 5 of the state, as described in algorithm 4

Note that when $CTX = 0$, the resulting state is exactly the same as AEGIS-256, as originally specified, without a context.

Algorithm 4 AEGIS-256 initialization with context

function INITIALIZE(CTX, K, IV)
  $S \leftarrow \{K_{256,0} \oplus IV_{256,0}, K_{256,1} \oplus IV_{256,1}, const_0, const_1, K_{256,0} \oplus const_1, K_{256,1} \oplus const_0\}$
  $i \leftarrow 0$
  while $i < 4$ **do**
    $S_3 \leftarrow S_3 \oplus Pad(CTX, 128)$
    $S_5 \leftarrow S_5 \oplus Pad(CTX, 128)$
    $S \leftarrow \mathbb{F}_{256}(S, K_{256,0})$
    $S_3 \leftarrow S_3 \oplus Pad(CTX, 128)$
    $S_5 \leftarrow S_5 \oplus Pad(CTX, 128)$
    $S \leftarrow \mathbb{F}_{256}(S, K_{256,1})$
    $S_3 \leftarrow S_3 \oplus Pad(CTX, 128)$
    $S_5 \leftarrow S_5 \oplus Pad(CTX, 128)$
    $S \leftarrow \mathbb{F}_{256}(S, IV_{256,0})$
    $S_3 \leftarrow S_3 \oplus Pad(CTX, 128)$
    $S_5 \leftarrow S_5 \oplus Pad(CTX, 128)$
    $S \leftarrow \mathbb{F}_{256}(S, IV_{256,1})$
    $i \leftarrow i + 1$
  end while
end function

4 The AEGIS-128X and AEGIS-256X modes

AEGIS absorbs the associated data and message with a rate r with $r = 256$ for AEGIS-128L and $r = 128$ for AEGIS-128 and AEGIS-256.

We define two new modes: AEGIS-128X[ν] and AEGIS-256X[ν], that absorb $r \cdot \nu$ bits per state update, spread over ν concurrent instances of the AEGIS-128L or AEGIS-256 encryption functions.

The associated data AD and plaintext P are split into interleaved blocks with a stride of $r \cdot \nu$ bits as they arrive. The last blocks are padded if necessary.

We first pad AD and P by adding trailing zero bits until they match the stride length:

$\hat{AD} = Pad(AD, r \cdot \nu)$
$\hat{P} = Pad(P, r \cdot \nu)$

$\hat{AD}$ is split into 128-bit blocks $\{\hat{AD}_0, \hat{AD}_1, \dots \hat{AD}_{\frac{|\hat{AD}|}{128}-1}\}$.

These blocks are interleaved to produce ν independent $\frac{|\hat{AD}|}{\nu}$ bit messages $\{\overline{AD}_0, \overline{AD}_1, \dots \overline{AD}_{\nu-1}\}$.

$\overline{AD}_0 = \hat{AD}_0 \| \hat{AD}_\nu \| \hat{AD}_{2\nu} \| \hat{AD}_{3\nu} \| \dots$
$\overline{AD}_1 = \hat{AD}_1 \| \hat{AD}_{\nu+1} \| \hat{AD}_{2\nu+1} \| \hat{AD}_{3\nu+1} \| \dots$
$\overline{AD}_2 = \hat{AD}_2 \| \hat{AD}_{\nu+2} \| \hat{AD}_{2\nu+2} \| \hat{AD}_{3\nu+2} \| \dots$
$\vdots$
$\overline{AD}_{\nu-1} = \hat{AD}_{\nu-1} \| \hat{AD}_{\nu+(nu-1)} \| \hat{AD}_{2\nu+(\nu-1)} \| \hat{AD}_{3\nu+(\nu-1)} \| \dots$

Similarly, $\hat{P}$ is split into ν independent $\frac{|\hat{P}|}{\nu}$ bit messages $\{\overline{P}_0, \overline{P}_1, \dots \overline{P}_{\nu-1}\}$.

4.1 AEGIS-128X

AEGIS-128X[ν] first encrypts and authenticates the plaintext and associated data fragments independently, producing ν ciphertexts $\{\overline{C}_0, \overline{C}_1, \dots \overline{C}_{\nu-1}\}$ and authentication tags $\{\overline{T}_0, \overline{T}_1, \dots \overline{T}_{\nu-1}\}$.

$\{\overline{C}_0, \overline{T}_0\} = Enc_{128L}(CTX \leftarrow 0, K, IV, \overline{A}_0, \overline{M}_0)$
$\{\overline{C}_1, \overline{T}_1\} = Enc_{128L}(CTX \leftarrow 1, K, IV, \overline{A}_1, \overline{M}_1)$
$\{\overline{C}_2, \overline{T}_2\} = Enc_{128L}(CTX \leftarrow 2, K, IV, \overline{A}_2, \overline{M}_2)$
$\vdots$
$\{\overline{C}_{\nu-1}, \overline{T}_{\nu-1}\} = Enc_{128L}(CTX \leftarrow \nu - 1, K, IV, \overline{A}_{\nu-1}, \overline{M}_{\nu-1})$

$\{\overline{C}_0, \overline{C}_1, \dots \overline{C}_{\nu-1}\}$ are deinterleaved to produce the final ciphertext:

$\hat{C} =$
$\overline{C}_{0,0} \| \overline{C}_{1,0} \| \overline{C}_{2,0} \| \dots \| \overline{C}_{(\nu-1),0} \|$
$\overline{C}_{0,1} \| \overline{C}_{1,1} \| \overline{C}_{2,1} \| \dots \| \overline{C}_{(\nu-1),1} \|$
$\overline{C}_{0,2} \| \overline{C}_{1,2} \| \overline{C}_{2,2} \| \dots \| \overline{C}_{(\nu-1),2} \| \dots$

$C = Trunc(\hat{C}, |P|)$

Finally, the AEGIS-128X[ν] authentication tag is the bit-wise exclusive OR of the AEGIS-128L authentication tags:

$T = \overline{T}_0 \oplus \overline{T}_1 \oplus \dots \oplus \overline{T}_{\nu-1}$

Note that AEGIS-128L and AEGIS-128X[1] are equivalent.

4.2 AEGIS-256X

AEGIS-256X[ν] uses the exact same interleaving technique as AEGIS-128X[ν] in order to process $r \cdot \nu$ bits per state update.

The only difference being that fragments are encrypted and authenticated using the AEGIS-256 encryption function instead of the AEGIS-128L one.

$$\{\overline{C}_0, \overline{T}_0\} = Enc_{256}(CTX \leftarrow 0, K, IV, \overline{A}_0, \overline{M}_0)$$
$$\{\overline{C}_1, \overline{T}_1\} = Enc_{256}(CTX \leftarrow 1, K, IV, \overline{A}_1, \overline{M}_1)$$
$$\{\overline{C}_2, \overline{T}_2\} = Enc_{256}(CTX \leftarrow 2, K, IV, \overline{A}_2, \overline{M}_2)$$
$$\vdots$$
$$\{\overline{C}_{\nu-1}, \overline{T}_{\nu-1}\} = Enc_{256}(CTX \leftarrow \nu - 1, K, IV, \overline{A}_{\nu-1}, \overline{M}_{\nu-1})$$

$\{\overline{C}_0, \overline{C}_1, \ldots \overline{C}_{\nu-1}\}$ are deinterleaved to produce the AEGIS-256X[ν] ciphertext:

$$\hat{C} = \overline{C}_{0,0} \| \overline{C}_{1,0} \| \overline{C}_{2,0} \| \ldots \| \overline{C}_{(\nu-1),0} \| \overline{C}_{0,1} \| \overline{C}_{1,1} \| \overline{C}_{2,1} \| \ldots \| \overline{C}_{(\nu-1),1} \| \overline{C}_{0,2} \| \overline{C}_{1,2} \| \overline{C}_{2,2} \| \ldots \| \overline{C}_{(\nu-1),2} \| \ldots$$

$$C = Trunc(\hat{C}, |P|)$$

Finally, the AEGIS-256X[ν] authentication tag is the bit-wise exclusive OR of the AEGIS-256 authentication tags:

$$T = \overline{T}_0 \oplus \overline{T}_1 \oplus \ldots \oplus \overline{T}_{\nu-1}$$

Note that AEGIS-256 and AEGIS-256X[1] are equivalent.

5 Rationale

The AEGIS security claims have the following requirements:

- Each key should be generated uniformly at random.
- Each key and IV pair should not be used to protect more than one message; and each key and IV pair should not be used with two different tag sizes.
- If verification fails, the decrypted plaintext and the wrong authentication tag should not be given as output.

AEGIS-128X[ν] and AEGIS-256X[ν] have the same requirements.

We'd like to emphasize that AEGIS-128X[ν] and AEGIS-256 are not new algorithms. They are modes, built on top of AEGIS-128L and AEGIS-256. designed to preserve the same security guarantees and requirements. Keys must be generated uniformly at random, key and IV pairs must not be reused, and for both variants the success rate of a forgery attack remains $2^{-|T|}$.

We did not include a parallel variant of AEGIS-128, as opposed to AEGIS-128L. AEGIS-128 trades performance for a smaller state size. But given that performance is the main justification for AEGIS-128X, AEGIS-128L feels like a more natural choice.

5.1 Implications of the AEGIS-128L context addition

$Enc_{128X}[\nu](K, IV, A, M)$ can be seen as ν evaluations of AEGIS-128L, on ν independent messages of the same length.

One way to satisfy the AEGIS-128L contract while reusing the key is to use distinct initialization vectors for each of the message fragments.

The parallelism degree ν, and thus the bounds of CTX, are limited by the hardware, and guaranteed to be small.

We could limit the AEGIS-128X[ν] IV size to $128 - log_2(\nu)$ bits (instead of 128 for AEGIS-128L), encoding the context in the remaining bits to create the IV used by the underlying AEGIS-128L function. That would be effectively AEGIS-128L, evaluated with independent messages, and distinct (K, IV) pairs. However, from an application perspective, a $128 - log_2(\nu)$ bit initialization vector would be unusual, and at odds with AEGIS-128L.

Ideally, we'd like AEGIS-128L to internally support $128 + log_2(\nu)$-bit initialization vectors: AEGIS-128X[ν] applications would use 128 bit initialization vectors, but the context could still be encoded to separate the parallel AEGIS-128L instances. To put it differently, we need to introduce a context with the same differential properties as the initialization vector.

In the proposed tweak to the initialization function, the context is added to the constants in blocks 3 and 7 AEGIS-128L of the initial state. The purpose of the constants (simply derived from the Fibonacci sequence) is to resist attacks exploiting the symmetry of the AES round function and of the overall AEGIS state.

Given its limited range, adding ν cannot turn them into weak constants, and doesn't alter any of the AEGIS-128L properties. Also note that ν is expected to be a hyperparameter, that an adversary cannot have control of.

Differential attacks could be a concern with the same (K, IV) pair used in different contexts. But in AEGIS-128L, there are 80 AES round functions (10 update steps) in the initialization function. In [STSI23], Shiraya at al. showed that the initialization phase of AEGIS-128L is secure against differential attacks after 3 update steps.

Furthermore, in order to prevent the difference in the state being eliminated completely in the middle of the initialization, the context difference is repeatedly injected into the state. This is consistent with how 128-bit initialization vectors are absorbed in AEGIS-128L.

The addition of a short context is thus unlikely to invalidate any of the current AEGIS-128L security claims.

The above security claims require a key and IV pair not to be used with different tag sizes. The AEGIS-128X[ν] construction guarantees that internal AEGIS-128L evaluations will always share the same tag size.

Note that the addition of a context to the AEGIS-128L initialization function could also be used to create a different initial state for different tag sizes, effectively increasing misuse resistance.

5.2 Implications of the AEGIS-256 context addition

The same observations apply to AEGIS-256X[ν] and its $Enc_{256X}[\nu]$ encryption function.

AEGIS-256 has a large initialization vector (256 bits). Exposing a shorter IV to applications while leveraging the remaining space for context separation would be reasonable. That would still allow applications to use random nonces with a negligible collision probability.

However, the large initialization vector has practical benefits, such as the ability to include a *protocol-specific* context with no overhead. In order to retain the same convenience as AEGIS-256 and for consistency with AEGIS-128X, our own context is added in a similar fashion.

In the AEGIS-256 initialization function, there are 96 round functions (16 update steps). According to the MILP-based evaluation from [STSI23], AEGIS-256 is secure against differential attacks after 6 update steps.

6 Implementation notes

Implementing AEGIS-128X[ν] and AEGIS-256X[ν] only requires trivial modifications to existing AEGIS-128L and AEGIS-256 implementations.

They apply the exact same operations as AEGIS-128L and AEGIS-256, to vectors of ν AES blocks instead of single blocks.

For example, with 256-bit registers, two AEGIS-128L states S and S' can be stored as:

$\{S_0, S'_0\}, \{S_1, S'_1\}, \{S_2, S'_2\}, \ldots \{S_7, S'_7\}$

This perfectly matches the AEGIS-128X[2] interleaved representation. In addition to the forward AES permutation, updating an AEGIS state only requires the bit-wise OR and AND operations. Even with wide registers, such operations are very efficient on any CPU with vector instructions.

On CPUs that don't implement vectorized versions of the AES permutation, AEGIS-128X[ν] and AEGIS-256X[ν] can be implemented in different ways:

- by emulating AES vector instructions. This is the easiest option, keeping the code close to hardware-accelerated versions.
- by evaluating $\{\overline{A}_0, \overline{A}_1, \overline{A}_2, \ldots \overline{A}_{\nu-1}\}$ and $\{\overline{C}_0, \overline{C}_1, \overline{C}_2, \ldots \overline{C}_{\nu-1}\}$ sequentially, with a periodic synchronization, for example after every memory page. This reduces cache-locality but also register pressure.

7 Performance evaluation

We implemented AEGIS-128X[2] and AEGIS-256X[2] using the Zig programming language. The code [Den23] is nearly identical to the reference implementations from the AEGIS specification [DSL23], with the AesBlock type extended to 2 AES blocks.

On Intel, AMD and ARM CPUs with vector registers but without AES vector instructions, AEGIS-128L and AEGIS-256 have better or comparable performance.

And unsurprinsingly, on CPUS without vector registers, AEGIS-128L and AEGIS-256 are consistently faster than their parallel counterparts.

However, on CPUs with the VAES instruction set such as an Intel Raptor Lake processor (table 1) or an AMD EPYC CPU (table 2, plot 7), AEGIS-128X[2] is almost twice as fast as AEGIS-128L with medium to large inputs.

With short inputs, the overhead of the parallel versions is small (plot 7). AEGIS-128X[2] is faster than AEGIS-128L as soon as the input size reaches 256 bytes, and AEGIS-256X[2] beats AEGIS-256 for inputs of size 128 or more.

Algorithm	Throughput
AEGIS-128X[2]	39781 MiB/s
AEGIS-128L	23863 MiB/s
AES128-GCM	10243 MiB/s

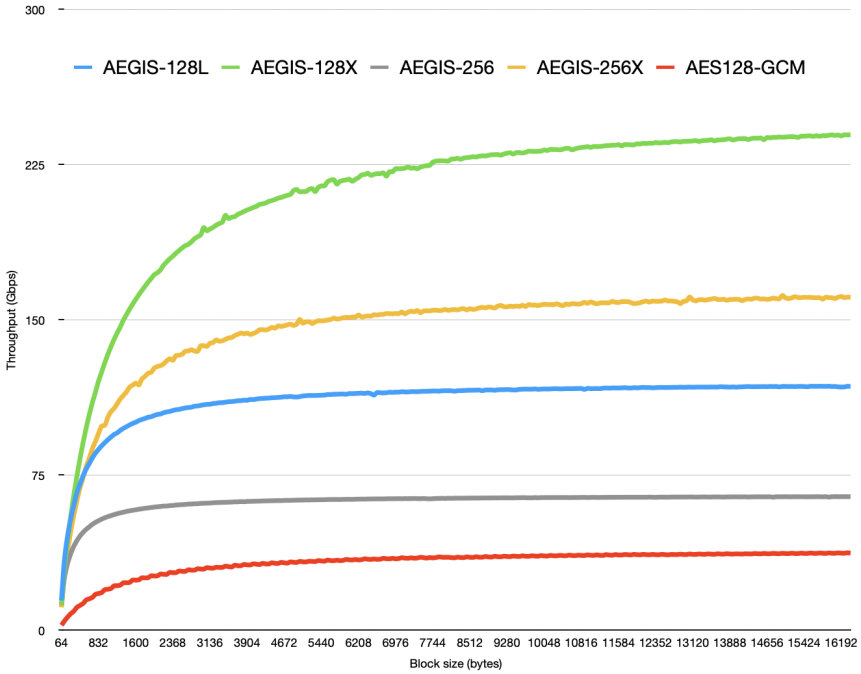
Table 1. Intel Core i9-13900K benchmark numbers (single core)

Algorithm	Throughput
AEGIS-128X[2]	239 Gbps
AEGIS-128L	118 Gbps
AES128-GCM	37 Gbps

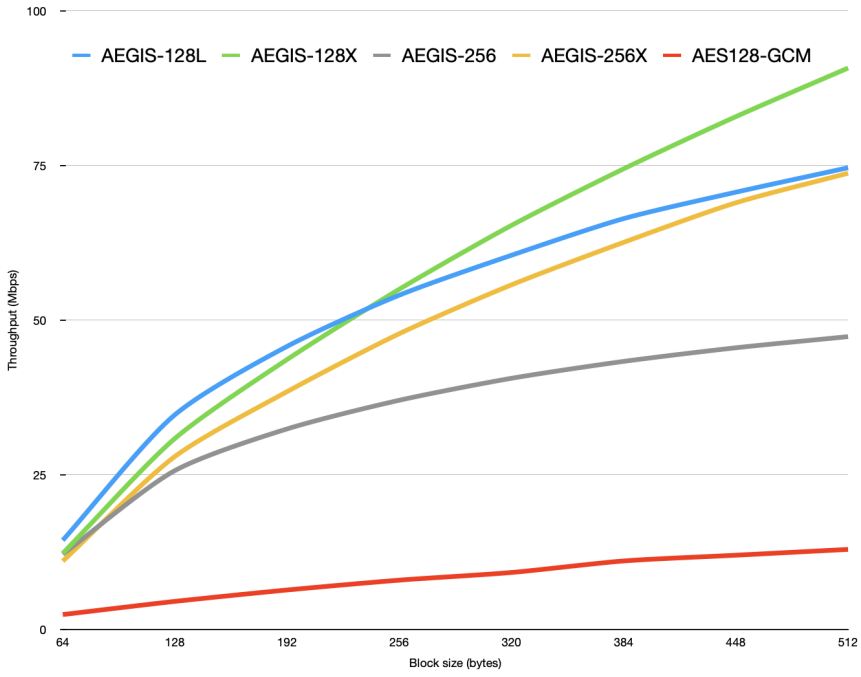
Table 2. AMD EPYC 7543 benchmark numbers (single core)

The benchmarked AES128-GCM implementation is the one from OpenSSL 3.1.0, while the AEGIS implementations are the reference code of the AEGIS specification, as well as our modified version to support for 128X variant.

AMD EPYC 7543 benchmark (single core)



AMD EPYC 7543 benchmark (single core, short inputs)



References

ADF+10. K. D. Akdemir, M. G. Dixon, W. K. Feghali, P. G. Fay, V. Gopal, J. Guilford, E. Ozturk, G. Wolrich, and R. Zohar. Breakthrough AES Performance with Intel ® AES New Instructions. 2010.

ARAR16. Ankele, Ralph, Ankele, and Robin. Software Benchmarking of the 2nd round CAESAR Candidates. Cryptology ePrint Archive, Report 2016/740, 2016. `https://eprint.iacr.org/2016/740`.

BLT15. A. Bogdanov, M. M. Lauridsen, and E. Tischhauser. Comb to Pipeline: Fast Software Encryption Revisited. In *FSE 2015*, *LNCS* 9054, pages 150–171. Springer, Heidelberg, March 2015.

Den23. F. Denis. AEGIS-128X and AEGIS-256X implementations. GitHub, 2023. `https://github.com/jedisct1/aegis-128x`.

DSL23. F. Denis, F. E. R. Scotoni, and S. Lucas. The AEGIS family of authenticated encryption algorithms. Internet-Draft draft-irtf-cfrg-aegis-aead-02, Internet Engineering Task Force, 2023. Work in Progress.

STSI23. T. Shiraya, N. Takeuchi, K. Sakamoto, and T. Isobe. MILP-based security evaluation for AEGIS/Tiaoxin-346/Rocca. *IET Information Security*, 2023.

VV18. S. Vaudenay and D. Vizár. Can Caesar Beat Galois? - Robustness of CAESAR Candidates Against Nonce Reusing and High Data Complexity Attacks. In *ACNS 18*, *LNCS* 10892, pages 476–494. Springer, Heidelberg, July 2018.

WP14. H. Wu and B. Preneel. AEGIS: A Fast Authenticated Encryption Algorithm. In *SAC 2013*, *LNCS* 8282, pages 185–201. Springer, Heidelberg, August 2014.

A Test vectors

Test vectors (in hexadecimal format) of AEGIS-128X and AEGIS-256X are given below.

A.1 AEGIS-128X[2]

```
Key:            000102030405060708090a0b0c0d0e0f
IV:             101112131415161718191a1b1c1d1e1f
AD:             (empty)
Plaintext:      (empty)
Ciphertext:     (empty)
128-bit tag:    d180aa6a90f29dedd6825b8c053f599b
```

```
Key:            000102030405060708090a0b0c0d0e0f
IV:             101112131415161718191a1b1c1d1e1f
AD:             0102030401020304
Plaintext:      0506070805060708050607080
Ciphertext:     985bac76fe1cefaa0b24d4d2
128-bit tag:    790b9a7342ee72e8b45202cd9c6195cf
```

A.2 AEGIS-256X[2]

```
Key:            000102030405060708090a0b0c0d0e0f
                101112131415161718191a1b1c1d1e1f
IV:             101112131415161718191a1b1c1d1e1f
                202122232425262728292a2b2c2d2e2f
AD:             (empty)
Plaintext:      (empty)
Ciphertext:     (empty)
128-bit tag:    ddbc7695b01623f82eee962f160bf267
```

```
Key:            000102030405060708090a0b0c0d0e0f
                101112131415161718191a1b1c1d1e1f
IV:             101112131415161718191a1b1c1d1e1f
                202122232425262728292a2b2c2d2e2f
AD:             0102030401020304
Plaintext:      0506070805060708050607080
Ciphertext:     a0b3f5b6b93db779c9d1b9de
128-bit tag:    fd2e93a6eb0b74dc30eb984fbec1d657
```