Adding more parallelism to the AEGIS authenticated encryption algorithms (v1.0)

Frank Denis

Fastly Inc.
fde@00f.net

Abstract. While the round function of the AEGIS authenticated encryption algorithms is highly parallelisable, their mode of operation is not.
We introduce two new modes to overcome that limitation: AEGIS-128X and AEGIS-256X. With minimal changes to existing implementations, they achieve a speedup of factor 4 over AEGIS-128L and AEGIS-256 on Intel CPUs with the VAES and AVX-512 instruction sets.

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.

Concurrent AES round instructions are required to fully utilize the AES pipelines. And the variant with the largest state, AEGIS-128L, updates eight 128-bit AES blocks concurrently. However, the mode of operation is similar to a duplex: after its initialization, the state is continuously updated by applying the round function to itself. 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" able to process multiple messages simultaneously.

The mode presented here also encrypts multiple messages simultaneously using the same cipher, but assumes that they 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 are the existing AEGIS algorithms, with a minor modification 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
$\lceil x \rceil$	: Ceiling operation; the smallest integer $\geq x$
$x \oplus y$	: Bit-wise exclusive OR
$\mathbb{F}_{128}(S, d_0, d_1)$	: AEGIS-128 state update function
$\mathbb{F}_{256}(S, d)$	: 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
$Enc_{128L}(CTX, K, IV, A, M)$	: The AEGIS-128L encryption function with context separation
$Enc_{256}(CTX, K, IV, A, M)$	: The AEGIS-256 encryption function with context separation
$Enc_{128X,\nu}(K, IV, A, M)$	: The AEGIS-128X parallel encryption function
$Enc_{256X,\nu}(K, IV, A, M)$	: The AEGIS-256X 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 i ($\frac{\lvert\hat{AD}\rvert}{\nu}$ 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 i ($\frac{\lvert\hat{C}\rvert}{\nu}$ 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 ($frac\lvert\hat{P}\rvert\nu$ 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.

3.1 Augmenting AEGIS-128L for context separation

AEGIS-128L defines the initial state S as a vector of eight AES blocks $\{S_0, S_1, \ldots S_7\}$ 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 ← {K_128 ⊕ IV_128, const_1, const_0, const_1, K_128 ⊕ IV_128, K_128 ⊕ const_0, K_128 ⊕ const_1, K_128 ⊕ const_0}
    i ← 0
    while i < 10 do
        S ← F_128(S, IV_128, K_128)
        i ← 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 3rd and 7th words 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 ← {K_128 ⊕ IV_128, const_1, const_0, const_1, K_128 ⊕ IV_128, K_128 ⊕ const_0, K_128 ⊕ const_1, K_128 ⊕ const_0}
    i ← 0
    while i < 10 do
        S_3 ← S_3 ⊕ Pad(CTX, 128)
        S_7 ← S_7 ⊕ Pad(CTX, 128)
        S ← F_128(S, IV_128, K_128)
        i ← 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 $\{S_0, S_1, \ldots S_5\}$ 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 3rd and 5th words 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-256 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 and AEGIS-256X, that absorb $r \cdot \nu$ bits per state update.

In the encryption function, the associated data AD and plaintext P are distributed in 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 $\{\hat{AD}_0, \hat{AD}_1, \ldots \hat{AD}_{\frac{|\hat{AD}|}{128}-1}\}$.

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

$$
\begin{aligned}
&\overline{AD}_0 = \hat{AD}_0 \| \hat{AD}_\nu \| \hat{AD}_{2\nu} \| \hat{AD}_{3\nu} \| \ldots \\
&\overline{AD}_1 = \hat{AD}_1 \| \hat{AD}_{\nu+1} \| \hat{AD}_{2\nu+1} \| \hat{AD}_{3\nu+1} \| \ldots \\
&\overline{AD}_2 = \hat{AD}_2 \| \hat{AD}_{\nu+2} \| \hat{AD}_{2\nu+2} \| \hat{AD}_{3\nu+2} \| \ldots \\
&\vdots \\
&\overline{AD}_{\nu-1} = \hat{AD}_{\nu-1} \| \hat{AD}_{\nu+(nu-1)} \| \hat{AD}_{2\nu+(\nu-1)} \| \hat{AD}_{3\nu+(\nu-1)} \| \ldots
\end{aligned}
$$

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

4.1 AEGIS-128X

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

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

$\{\overline{C}_0, \overline{C}_1, \ldots \overline{C}_{\nu-1}\}$ are deinterleaved to produce the AEGIS-128X ciphertext:

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

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

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

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

Note that AEGIS-128L is a specific instance of AEGIS-128X with $\nu = 1$.

4.2 AEGIS-256X

AEGIS-256X uses the exact same interleaving technique as AEGIS-128X 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 function instead of AEGIS-128L.

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

$\{\overline{C}_0, \overline{C}_1, \ldots \overline{C}_{\nu-1}\}$ are deinterleaved to produce the AEGIS-128X 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-256X authentication tag is the bit-wise exclusive OR of the AEGIS-256 authentication tags:

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

Note that AEGIS-256 is a specific instance of AEGIS-256X with $\nu = 1$.

5 Security

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 and AEGIS-256X have the same requirements.

$Enc_{128X,\nu}(K, IV, A, M)$ can be seen as ν evaluations of AEGIS-128L, on ν independent messages of length $\lceil \frac{|P|}{\nu} \rceil$ bits.

In order to satisfy the AEGIS-128L contract, we should either derive distinct keys for each of these messages, or use distinct initialization vectors.

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 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 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.

The main concern with the same (K, IV) pair used in different contexts are differential attacks.

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 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.

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

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.

We'd like to emphasize that AEGIS-128X 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 $2^{-|T|}$.

6 Implementation notes

Implementing AEGIS-128X and AEGIS-256X 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 AES blocks instead of single blocks.

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

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

This perfectly matches the AEGIS-128X 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 and AEGIS-256X can be implemented in two different ways:

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

7 Performance evaluation

The AEGIS-128X and AEGIS-256X inputs are transformed into evenly distributed fragments. As a result, they only yield a performance improvement over their underlying primitives when $|A| > r$ or $|P| > r$.

We implemented AEGIS-128X and AEGIS-256X using the Zig programming language. The code is nearly identical to the reference implementations from the AEGIS specification [DSL23], with the AesBlock type extended to 2 and 4 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 an Intel Raptor Lake processor (Core i9-13900K) with the VAES instruction set, AEGIS-128X with a parallelism degree of 2 is almost twice as fast as AEGIS-128L with large inputs.

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

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.

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.

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.

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.