AEGIS: A Fast Authenticated Encryption Algorithm[⋆] (Full Version)

Hongjun Wu[1], Bart Preneel[2]

[1] School of Physical and Mathematical Sciences
Nanyang Technological University
wuhj@ntu.edu.sg
[2] Dept. Elektrotechniek-ESAT/COSIC
KU Leuven and iMinds, Ghent
bart.preneel@esat.kuleuven.be

Abstract. This paper introduces a dedicated authenticated encryption algorithm AEGIS; AEGIS allows for the protection of associated data which makes it very suitable for protecting network packets. AEGIS-128L uses eight AES round functions to process a 32-byte message block (one step). AEGIS-128 uses five AES round functions to process a 16-byte message block (one step); AES-256 uses six AES round functions. The security analysis shows that these algorithms offer a high level of security. On the Intel Sandy Bridge Core i5 processor, the speed of AEGIS-128L, AEGIS-128 and AEGIS-256 is around 0.48, 0.66 and 0.7 clock cycles/byte (cpb) for 4096-byte messages, respectively. This is substantially faster than the AES CCM, GCM and OCB modes.

Key words: Authenticated encryption, AEGIS, AES-NI

1 Introduction

The protection of a message typically requires the protection of both confidentiality and authenticity. There are two main approaches to authenticate and encrypt a message. One approach is to treat the encryption and authentication separately. The plaintext is encrypted with a block cipher or stream cipher, and a MAC algorithm is used to authenticate the ciphertext. For example, we may apply AES [17] in CBC mode [18] to the plaintext, then apply AES-CMAC [22] (or Pelican MAC [6] or HMAC [19]) to the ciphertext to generate an authentication tag. This approach is relatively easy to analyze since the security of authentication and encryption can be analyzed almost separately. Bellare and Namprempre have performed a detailed analysis of this type of authenticated encryption for randomized encryption [2]. Another approach is to apply an integrated authenticated encryption algorithm to the message; one can expect that

[⋆] The original paper was published at Selected Areas in Cryptography (SAC 2013). AEGIS-128L is introduced in this full version.

this is more efficient since authentication and encryption can share part of the computation.

There are three approaches to design an integrated authenticated encryption algorithm. The first approach is to use a block cipher in a special mode (the block cipher is treated as a black box). The research on this approach started about ten years ago [9, 12, 14]. There are now two NIST recommended modes of operation for authenticated encryption, namely, CCM [20] and GCM [21]. OCB [24, 25, 15] is a widely known authenticated encryption mode, and OCB2 is an ISO standard. The second approach is to use a stream cipher (the stream cipher is treated as a black box). The keystream is divided into two parts: one part for encryption and another part for authentication. A typical example of this approach is Grain-128a [1]. The third approach is to design dedicated authenticated encryption algorithms. In this approach, a message is used to update the state of the cipher, and message authentication can be achieved almost for free. Two examples of this approach are Helix [8] and Phelix [26]. The attack against Phelix [27] shows that it is unlikely that this type of authenticated encryption algorithm can withstand nonce-reuse attacks if it requires much less computation than a block cipher.

In this paper, we propose a dedicated authenticated encryption algorithm AEGIS following the third approach above. AEGIS is constructed from the AES encryption round function (not the last round). AEGIS-128L uses eight AES round functions to process a 32-byte message block (one step). AEGIS-128 processes a 16-byte message block with 5 AES round functions, and AEGIS-256 uses 6 AES round functions. The computational cost of AEGIS is about half that of AES. AEGIS is very fast. On the Intel Sandy Bridge processor Core-i5, the encryption speeds of AEGIS-128L, AEGIS-128 and AEGIS-256 are about 0.48cpb, 0.66 cpb and 0.70 cpb, respectively. The speeds of AEGIS-128L is much faster than that of AES in counter (CTR) mode, and are about 8 times that of AES encryption in CBC mode. AEGIS offers a very high security. As long as the nonce is not reused, it is impossible to recover the AEGIS state and key faster than exhaustive key search (under the assumption that a 128-bit authentication tag is used, and the forgery attack cannot be repeated for the same key for more than 2^{128} times). AEGIS is suitable for network communication since it is straightforward to use AEGIS to protect a packet while leaving the packet header (associated data) unencrypted.

This paper is organized as follows. The operations, variables and functions are introduced in Sect. 2. The specifications of AEGIS-128, AEGIS-256 and AEGIS-128L are given in Sect. 3, Sect. 4 and Sect. 5, respectively. Section 6 gives the security analysis of AEGIS-128, AEGIS-256 and AEGIS-128L. The software performance of AEGIS is given in Sect. 7. The design rationale is given in Sect. 8. Section 9 concludes this paper.

2 Operations, Variables and Functions

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

2.1 Operations

The following operations are used in AEGIS:

$\oplus$	: bit-wise exclusive OR
$\&$	: bit-wise AND
$\|$	: concatenation
$\lceil x \rceil$	: ceiling operation, $\lceil x \rceil$ is the smallest integer not less than x

2.2 Variables and constants

The following variables and constants are used in AEGIS:

AD	: associated data (this data will not be encrypted or decrypted).
AD_i	: a 16-byte associated data block (the last block may be a partial block).
$adlen$	: bit length of the associated data with $0 \le adlen < 2^{64}$.
C	: ciphertext.
C_i	: a 16-byte ciphertext block (the last block may be a partial block).
$const$	: a 32-byte constant in the hexadecimal format; $const = 00 \parallel 01 \parallel 01 \parallel 02 \parallel 03 \parallel 05 \parallel 08 \parallel 0d \parallel 15 \parallel 22 \parallel 37 \parallel 59 \parallel 90 \parallel e9 \parallel 79 \parallel 62 \parallel db \parallel 3d \parallel 18 \parallel 55 \parallel 6d \parallel c2 \parallel 2f \parallel f1 \parallel 20 \parallel 11 \parallel 31 \parallel 42 \parallel 73 \parallel b5 \parallel 28 \parallel dd$. This is the Fibonacci sequence modulo 256.
$const_0$	: first 16 bytes of $const$.
$const_1$	: last 16 bytes of $const$.
IV_{128}	: 128-bit initialization vector of AEGIS-128.
IV_{256}	: 256-bit initialization vector of AEGIS-256.
$IV_{256,0}$	: first half of IV_{256} .
$IV_{256,1}$	: second half of IV_{256} .
K_{128}	: 128-bit key of AEGIS-128.
K_{256}	: 256-bit key of AEGIS-256.
$K_{256,0}$	: first half of K_{256} .
$K_{256,1}$	: second half of K_{256} .
$msglen$	: bit length of the plaintext/ciphertext with $0 \le msglen < 2^{64}$.
m_i	: a 16-byte data block.
P	: plaintext.
P_i	: a 16-byte plaintext block (the last block may be a partial block).
S_i	: state at the beginning of the ith step.
$S_{i,j}$	: j-th 16-byte element of the state S_i . For AEGIS-128, $0 \le j \le 4$; for AEGIS-256, $0 \le j \le 5$; for AEGIS-128L, $0 \le j \le 7$.
T	: authentication tag.
t	: bit length of the authentication tag with $64 \le t \le 128$.
u	: $u = \lceil \frac{adlen}{128} \rceil$.
v	: $v = \lceil \frac{msglen}{128} \rceil$.
u_L	: $u_L = \lceil \frac{adlen}{256} \rceil$.
v_L	: $v_L = \lceil \frac{msglen}{256} \rceil$.

2.3 Functions

The AES encryption round function (not the last round) is used in AEGIS:

$AESRound(A, B)$: A is the 16-byte state, B is the 16-byte round key. This function mapping 2 16-byte inputs to a 16-byte output can be implemented efficiently on recent x86 processors using the AES instruction `_mm_aesenc_si128(A, B)`, where A and B are two 128-bit integers `__m128i`.

3 AEGIS-128

In this section, we describe AEGIS-128. With a 128-bit key and a 128-bit initialization vector, AEGIS-128 encrypts and authenticates a message. The associated data length and the plaintext length are less than 2^{64} bits. The authentication tag length is less than or equal to 128 bits. We strongly recommend the use of a 128-bit tag.

3.1 The state update function of AEGIS-128

The state update function updates the 80-byte state S_i with a 16-byte message block m_i. $S_{i+1} = \texttt{StateUpdate128}(S_i, m_i)$ is given as follows:

$$\begin{aligned}
S_{i+1,0} &= AESRound(S_{i,4}, S_{i,0} \oplus m_i);\\
S_{i+1,1} &= AESRound(S_{i,0}, S_{i,1});\\
S_{i+1,2} &= AESRound(S_{i,1}, S_{i,2});\\
S_{i+1,3} &= AESRound(S_{i,2}, S_{i,3});\\
S_{i+1,4} &= AESRound(S_{i,3}, S_{i,4});
\end{aligned}$$

The state update function is shown in Fig. 1 :

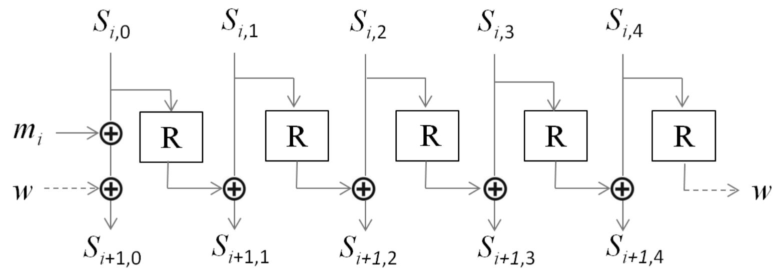

Fig. 1. The state update function of AEGIS-128. R indicates the AES encryption round function without XORing with the round key and w is a temporary 16-byte word.

3.2 The initialization of AEGIS-128

The initialization of AEGIS-128 consists of loading the key and IV into the state, and running the cipher for 10 steps with the key and IV being used as message.

1. Load the key and IV into the state as follows:

$$\begin{aligned} &S_{-10,0} = K_{128} \oplus IV_{128}; \\ &S_{-10,1} = const_1; \\ &S_{-10,2} = const_0; \\ &S_{-10,3} = K_{128} \oplus const_0; \\ &S_{-10,4} = K_{128} \oplus const_1; \end{aligned}$$

2. For $i = -5$ to -1, $m_{2i} = K_{128}$; $m_{2i+1} = K_{128} \oplus IV_{128}$;

3. For $i = -10$ to -1, $S_{i+1} = \texttt{StateUpdate128}(S_i, m_i)$;

3.3 Processing the authenticated data

After the initialization, the associated data AD is used to update the state.

1. If the last associated data block is not a full block, use 0 bits to pad it to 128 bits, and the padded full block is used to update the state. Note that if $adlen = 0$, the state will not be updated.
2. For $i = 0$ to $\lceil \frac{adlen}{128} \rceil - 1$, we update the state:

$$S_{i+1} = \texttt{StateUpdate128}(S_i, AD_i) ;$$

3.4 The encryption of AEGIS-128

After processing the associated data, at each step of the encryption, a 16-byte plaintext block P_i is used to update the state, and P_i is encrypted to C_i.

1. If the last plaintext block is not a full block, use 0 bits to pad it to 128 bits, and the padded full block is used to update the state. But only the partial block is encrypted. Note that if $msglen = 0$, the state will not get updated, and there is no encryption.

2. Let $u = \lceil \frac{adlen}{128} \rceil$ and $v = \lceil \frac{msglen}{128} \rceil$. For $i = 0$ to $v - 1$, we perform encryption and update the state:

$$\begin{aligned} C_i &= P_i \oplus S_{u+i,1} \oplus S_{u+i,4} \oplus (S_{u+i,2} \& S_{u+i,3}) ; \\ S_{u+i+1} &= \texttt{StateUpdate128}(S_{u+i}, P_i) ; \end{aligned}$$

3.5 The finalization of AEGIS-128

After encrypting all the plaintext blocks, we generate the authentication tag using seven more steps. The length of the associated data and the length of the message are used to update the state.

1. Let $tmp = S_{u+v,3} \oplus (adlen \parallel msglen)$, where *adlen* and *msglen* are represented as 64-bit integers.

2. For $i = u + v$ to $u + v + 6$, we update the state:

$$S_{i+1} = \texttt{StateUpdate128}(S_i, tmp) \ ;$$

3. We generate the authentication tag from the state S_{u+v+7} as follows:
$T' = \bigoplus_{i=0}^{4} S_{u+v+7,i}$;

The authentication tag T consists of the first t bits of T'.

3.6 The decryption and verification of AEGIS-128

The exact values of key size, IV size, and tag size should be known to the decryption and verification processes. The decryption starts with the initialization and the processing of authenticated data. Then the ciphertext is decrypted as follows:

1. If the last ciphertext block is not a full block, decrypt only the partial ciphertext block. The partial plaintext block is padded with 0 bits, and the padded full plaintext block is used to update the state.

2. For $i = 0$ to $v - 1$, we perform decryption and update the state.

$$\begin{aligned} P_i &= C_i \oplus S_{u+i,1} \oplus S_{u+i,4} \oplus (S_{u+i,2} \& S_{u+i,3}) \ ; \\ S_{u+i+1} &= \texttt{StateUpdate128}(S_{u+i}, P_i) \ ; \end{aligned}$$

The finalization in the decryption process is the same as that in the encryption process. We emphasize that if the verification fails, the ciphertext and the newly generated authentication tag should not be given as output; otherwise, the state of AEGIS-128 is vulnerable to known-plaintext or chosen-ciphertext attacks (using a fixed IV). This requirement also applies to AEGIS-256.

4 AEGIS-256

In this section, we describe AEGIS-256. With a 256-bit key and a 256-bit initialization vector, AEGIS-256 encrypts and authenticates a message. The associated data length and the plaintext length are less than 2^{64} bits. The authentication tag length is less than or equal to 128 bits. We strongly recommend the use of a 128-bit tag.

4.1 The state update function of AEGIS-256

The state update function updates the 96-byte state S_i with a 16-byte message block m_i. $S_{i+1} = \texttt{StateUpdate256}(S_i, m_i)$ is illustrated as follows:

$$\begin{aligned}
S_{i+1,0} &= AESRound(S_{i,5}, S_{i,0} \oplus m_i);\\
S_{i+1,1} &= AESRound(S_{i,0}, S_{i,1});\\
S_{i+1,2} &= AESRound(S_{i,1}, S_{i,2});\\
S_{i+1,3} &= AESRound(S_{i,2}, S_{i,3});\\
S_{i+1,4} &= AESRound(S_{i,3}, S_{i,4});\\
S_{i+1,5} &= AESRound(S_{i,4}, S_{i,5});
\end{aligned}$$

4.2 The initialization of AEGIS-256

The initialization of AEGIS-256 consists of loading the key and IV into the state, and running the cipher for 16 steps with the key and IV being used as message.

1. Load the key and IV into the state as follows:
$$\begin{aligned}
S_{-16,0} &= K_{256,0} \oplus IV_{256,0};\\
S_{-16,1} &= K_{256,1} \oplus IV_{256,1};\\
S_{-16,2} &= const_1;\\
S_{-16,3} &= const_0;\\
S_{-16,4} &= K_{256,0} \oplus const_0;\\
S_{-16,5} &= K_{256,1} \oplus const_1;
\end{aligned}$$
2. For $i = -4$ to -1,
$$\begin{aligned}
m_{4i} &= K_{256,0}\ ;\\
m_{4i+1} &= K_{256,1}\ ;\\
m_{4i+2} &= K_{256,0} \oplus IV_{256,0}\ ;\\
m_{4i+3} &= K_{256,1} \oplus IV_{256,1}\ .
\end{aligned}$$
3. For $i = -16$ to -1, $S_{i+1} = \texttt{StateUpdate256}(S_i, m_i)$;

4.3 Processing the authenticated data

After the initialization, the associated data AD is used to update the state.

1. If the last associated data block is not a full block, use 0 bits to pad it to 128 bits, and the padded full block is used to update the state. Note that if $adlen = 0$, the state will not get updated.
2. For $i = 0$ to $\lceil \frac{adlen}{128} \rceil - 1$, we update the state.

$$S_{i+1} = \texttt{StateUpdate256}(S_i, AD_i)\ ;$$

4.4 The encryption of AEGIS-256

After processing the associated data, at each step of the encryption, a 16-byte plaintext block P_i is used to update the state, and P_i is encrypted to C_i.

1. If the last plaintext block is not a full block, use 0 bits to pad it to 128 bits, and the padded full block is used to update the state. But only the partial block is encrypted. Note that if $msglen = 0$, the state will not get updated, and there is no encryption.

2. Let $u = \lceil \frac{adlen}{128} \rceil$ and $v = \lceil \frac{msglen}{128} \rceil$. For $i = 0$ to $v - 1$, we perform encryption and update the state:

$$\begin{aligned} C_i &= P_i \oplus S_{u+i,1} \oplus S_{u+i,4} \oplus S_{u+i,5} \oplus (S_{u+i,2} \& S_{u+i,3}) \ ; \\ S_{u+i+1} &= \texttt{StateUpdate256}(S_{u+i}, P_i) \ ; \end{aligned}$$

4.5 The finalization of AEGIS-256

After encrypting all the plaintext blocks, we generate the authentication tag using seven more steps. The length of the associated data and the length of the message are used to update the state.

1. Let $tmp = S_{u+v,3} \oplus (adlen \parallel msglen)$, where $adlen$ and $msglen$ are represented as 64-bit integers.

2. For $i = u + v$ to $u + v + 6$, we update the state:

$$S_{i+1} = \texttt{StateUpdate256}(S_i, tmp) \ ;$$

3. We generate the authentication tag from the state S_{u+v+7} as follows:
$$T' = \bigoplus_{i=0}^{5} S_{u+v+7,i} \ ;$$

The authentication tag T consists of the first t bits of T'.

5 AEGIS-128L

In this section, we describe AEGIS-128L with 128-byte state. Its key, IV and tag sizes are the same as that of AEGIS-128. AEGIS-128L encrypts and authenticates a message with length less than 2^{64} bits.

5.1 The state update function of AEGIS-128L

The state update function AEGIS-128L updates the 128-byte state S_i with two 16-byte message blocks m_a and m_b. $S_{i+1} = \texttt{StateUpdate128L}(S_i, m_a, m_b)$ is illustrated as follows:

$$
\begin{aligned}
S_{i+1,0} &= AESRound(S_{i,7}, S_{i,0} \oplus m_a)\,;\\
S_{i+1,1} &= AESRound(S_{i,0}, S_{i,1});\\
S_{i+1,2} &= AESRound(S_{i,1}, S_{i,2});\\
S_{i+1,3} &= AESRound(S_{i,2}, S_{i,3});\\
S_{i+1,4} &= AESRound(S_{i,3}, S_{i,4} \oplus m_b)\,;\\
S_{i+1,5} &= AESRound(S_{i,4}, S_{i,5});\\
S_{i+1,6} &= AESRound(S_{i,5}, S_{i,6});\\
S_{i+1,7} &= AESRound(S_{i,6}, S_{i,7});
\end{aligned}
$$

5.2 The initialization of AEGIS-128L

The initialization of AEGIS-128L consists of loading the key and IV into the state, and running the cipher for 10 steps with the key and IV being used as message.

1. Load the key and IV into the state as follows:

$$
\begin{aligned}
S_{-10,0} &= K_{128} \oplus IV_{128};\\
S_{-10,1} &= const_1;\\
S_{-10,2} &= const_0;\\
S_{-10,3} &= const_1;\\
S_{-10,4} &= K_{128} \oplus IV_{128};\\
S_{-10,5} &= K_{128} \oplus const_0;\\
S_{-10,6} &= K_{128} \oplus const_1;\\
S_{-10,7} &= K_{128} \oplus const_0;
\end{aligned}
$$

2. For $i = -10$ to -1, $S_{i+1} = \texttt{StateUpdate128L}(S_i, IV_{128}, K_{128})$.

5.3 Processing the authenticated data

After the initialization, the associated data AD is used to update the state.

1. If the length of associated data is not a multiple of 256 bits, use 0 bits to pad it to $\lceil \frac{adlen}{256} \rceil \times 256$ bits, and the padded associated data is used to update the state. Note that if $adlen = 0$, the state will not get updated.
2. For $i = 0$ to $\lceil \frac{adlen}{256} \rceil - 1$, we update the state.

$$S_{i+1} = \texttt{StateUpdate128L}(S_i, AD_{2i}, AD_{2i+1})\,;$$

5.4 The encryption of AEGIS-128L

After the initialization, in every step of the encryption, two 16-byte plaintext blocks P_{2i} and P_{2i+1} are used to update the state S_i to obtain the state S_{i+1}, and the plaintext blocks get encrypted.

1. If the size of the message is not a multiple of 256 bits, use 0 bits to pad it to $\lceil \frac{msglen}{256} \rceil \times 256$ bits.

2. Let $u_L = \lceil \frac{adlen}{256} \rceil$, $v_L = \lceil \frac{msglen}{256} \rceil$. For $i = 0$ to $v_L - 1$, we update the state and perform encryption.

$$\begin{aligned} C_{2i} &= P_{2i} \oplus S_{u_L+i,1} \oplus S_{u_L+i,6} \oplus (S_{u_L+i,2} \& S_{u_L+i,3}) \ ; \\ C_{2i+1} &= P_{2i+1} \oplus S_{u_L+i,2} \oplus S_{u_L+i,5} \oplus (S_{u_L+i,6} \& S_{u_L+i,7}) \ ; \\ S_{u_L+i+1} &= \texttt{StateUpdate128L}(S_{u_L+i}, P_{2i}, P_{2i+1}) \ ; \end{aligned}$$

5.5 The finalization of AEGIS-128L

After encrypting all the plaintext blocks, we generate the authentication tag using seven more steps. The message being used at this stage is part of the state at the end of the encryption, together with the length of the associated data and the length of the message.

1. Let $tmp = S_{u_L+v_L,2} \oplus (adlen \parallel msglen)$, where $adlen$ and $msglen$ are represented as 64-bit integers.

2. For $i = u_L + v_L$ to $u_L + v_L + 6$, we update the state:
   $S_{i+1} = \texttt{StateUpdate128L}(S_i, tmp, tmp)$.

3. We generate the authentication tag from the state $S_{u_L+v_L+7}$ as follows:
   $T' = \bigoplus_{i=0}^{7} S_{u_L+v_L+7,i}$.

The authentication tag T is the first t bits of T' .

6 The Security of AEGIS

The following requirements should be satisfied in order to use AEGIS securely.

1. Each key should be generated uniformly at random.
2. 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.
3. If verification fails, the decrypted plaintext and the wrong authentication tag should not be given as output.

If the above requirements are satisfied, we have the following security claims:

Claim 1. The success rate of a forgery attack is 2^{-t}, where t is the tag size. If the forgery attack is repeated n times, the success rate of a forgery attack is about $n \times 2^{-t}$.

Claim 2. The state and key cannot be recovered faster than exhaustive key search if the forgery attack is not successful. We recommend the use of a 128-bit tag size for AEGIS in order to resist repeated forgery attacks. (Note that with 128-bit tag, the state of AEGIS-256 can be recovered faster than exhaustive key search if a forgery attack is repeated for about 2^{128} times for the same key and IV pair.)

6.1 The security of the initialization

A difference in IV is the main threat to the security of the initialization of AEGIS. A difference in IV would eventually propagate into the ciphertexts, and thus it is possible to apply a differential attack against AEGIS. In AEGIS-128, there are 50 AES round functions (10 steps) in the initialization. If there is a difference in IV, the difference would pass through more than 10 AES round functions. In AEGIS-256, there are 96 AES round functions (16 steps) in the initialization. If there is a difference in IV, the difference would pass through more than 16 AES round functions. In AEGIS-128L, there are 80 AES round functions (10 steps) in the initialization. If there is a difference in IV, the difference would pass through more than 20 AES round functions. Furthermore, in order to prevent the difference in the state being eliminated completely in the middle of the initialization, we inject the IV difference repeatedly into the state (5, 8 and 10 times into the state of AEGIS-128, AEGIS-256 and AEGIS-128L, respectively). We expect that a differential attack against the initialization would be more expensive than exhaustive key search.

6.2 The security of the encryption process

We emphasize here that AEGIS encryption is a stream cipher with a large state which is updated continuously. The attacks against a block cipher cannot be applied directly to AEGIS. The state update function involves five AES round functions in AEGIS-128, and six AES round functions in AEGIS-256. We should ensure that IV is not reused for the same key; otherwise, the states of AEGIS can be recovered easily with either known-plaintext attacks or chosen plaintext attacks. For example, if we re-use an IV and inject a difference into P_i, the difference would propagate into C_{i+2}, and part of the state can be attacked by analyzing the difference pair $(\Delta P_i, \Delta C_{i+2})$. If an authenticated encryption algorithm is secure for re-used IVs, we expect that such an algorithm can only be as fast as a block cipher, as pointed out in [27]. This can be argued as follows: once an IV is re-used, the attacks that are relevant for a block cipher can be applied to attack the state.

Statistical Attacks. If the IV is used only once for each key, it is impossible to apply a differential attack to the encryption process. It is extremely difficult to apply a linear attack (or correlation attack) to recover the secret state since the state of AEGIS is updated in a nonlinear way. In general, it would be difficult to apply any statistical attack to recover the secret state due to the nonlinear state update function (the statistical correlation between any two states vanishes quickly as the distance between them increases).

LEX [3, 4] is an AES-based stream cipher that generates keystream from part of the state. We would like to mention here that AEGIS is not vulnerable to the attack against LEX [7]. There is a fundamental reason why LEX is vulnerable to a statistical attack while AEGIS is not: the round keys used in LEX are fixed, while the whole state of AEGIS is updated continuously in a nonlinear way.

6.3 The security of message authentication

There are two main approaches to attack a MAC algorithm. One approach is to recover the secret key or secret state, another approach is to introduce/detect an internal state collision. Besides these two approaches, when we analyze the security of message authentication, we need to consider that the AEGIS encryption may affect the security of message authentication.

Recovering key or state. From Sect. 6.1, we expect that the secret key cannot be recovered faster than exhaustive search by attacking the initialization. From Sect. 6.2, we expect that the state cannot be recovered faster than exhaustive search by attacking the encryption process if the IV is used only once. Similarly, we expect that the state cannot be recovered faster than exhaustive search by attacking the tag generation process if IV is not reused.

An attacker can still inject a difference into the state in the tag verification process and obtain the decrypted plaintext if the forgery attack is allowed to be repeated for multiple times for the same key and IV pair. In a forgery attack, the decrypted plaintext is known to the attacker with probability 2^{-t} (if the verification is successful). It becomes possible to recover the state if the forgery attack is repeated many times. We recommend the use of 128-bit tag so that recovering the state requires at least 2^{128} forgery attempts.

The security level of the AEGIS-256 state is only 128 bits with a 128-bit tag (if we consider that a forgery attack becomes successful). However, we believe that repeating the forgery attack for around 2^{128} times to recover a state is impractical.

Internal collisions. A powerful attack against MAC is to introduce and detect internal collisions. A general approach based on the birthday attack was given by Preneel and van Oorschot [23]: an internal collision can be detected after a key is used to generate the authentication tags of about $2^{n/2}$ chosen messages, where n is the state size and tag size in bits. The internal collision can be exploited to forge the tags of new messages. The birthday attack was later applied to other MAC algorithms [28]. AEGIS resists this type of attacks due to its large state size. Another approach to introduce internal collision is through differential cryptanalysis. Suppose that the difference cancellation in the state occurs with probability 2^{-a}; then we can detect an internal collision after a secret key is used to generate the tags of those 2^a message pairs. The resulting internal collision can be used to forge the tags of new messages.

An attacker can inject a difference into the state in the decryption and tag verification process by modifying the ciphertext. However, AEGIS provides a large security margin against this type of attack since differences are introduced into a large state. The security of AEGIS against forgery attack is stronger than that of Pelican MAC when the message or the tag gets modified. In Pelican MAC, four AES round functions are used to process each 16-byte message block; while in AEGIS, at least four AES round functions are used. Furthermore, the state size of AEGIS-128 is at least 5 times that of Pelican MAC, and it becomes much more difficult to eliminate the difference in the large state. A simple description of our analysis is given below. We notice that the first difference being injected into ciphertext would pass through five round functions without being affected by another ciphertext difference in AEGIS-128, and there are at least 26 active Sboxes being involved. If we consider only a single differential path, the probability of the difference cancellation in the state is less than $2^{-6\times 26} = 2^{-156}$. Thus generating a state collision in the verification process requires at least 2^{156} modifications to the ciphertext. Note that the differential attack here is slightly different from that against block cipher since the AEGIS verification process would guarantee that each forgery attack generates only one useful difference pair (the failed forgery attacks would not give outputs). It shows that AEGIS-128 is strong against forgery attack when the ciphertext or tag gets modified. Multiple differential paths would not have a significant effect on the forgery attack here, since each differential path has to cancel its own differences being left in the state. Attacking AEGIS-256 is more difficult since it involves a larger state and more AES round functions. The security of AEGIS-128L against forgery attack is slightly weaker than AEGIS-128 since a difference passes through at least four AES rounds. However, a forgery attack against AEGIS-128L still requires at least 2^{150} modifications to the ciphertext. Note that our analysis above is very conservative since when a difference passes through five AES round functions, the difference would be injected into each 16-byte element in the state.

We now analyze whether the noninvertible AEGIS state update function affects the security of the authentication of AEGIS. In AEGIS, a difference in the state could be eliminated even if there is no difference being introduced to cancel it. However, it would only happen if the difference in every 16-byte element is able to eliminate the difference in the next element after passing through an AES round function. It means that at least 26 active Sboxes are involved in this difference elimination process in AEGIS-128, and generating these particular differences in the state involves more than 26 additional active Sboxes. We consider that this type of weak state difference has a negligible effect on the security of the authentication of AEGIS.

The analysis given above shows that the authentication of AEGIS is very strong.

6.4 Other attacks

There are weak states in AEGIS. In one type of weak states, all the 16-byte elements in a state are equal: consequently all the 16-byte elements in the next state would be equal (if the message block is 0). However, there are only 2^{128} such states, so this type of weak state appears with probabilities 2^{-512}, 2^{-640} and 2^{-896} for AEGIS-128, AEGIS-256 and AEGIS-128L, respectively. In another type of weak states, the four columns in each 16-byte element are equal and every 16-byte element has such a property: in this case, the same property would appear in the next state (if the message block also has such a property). However, there are only $2^{32\times 5} = 2^{160}$ such states in AEGIS-128, $2^{32\times 6} = 2^{192}$ such states in

AEGIS-256, $2^{32\times 8} = 2^{256}$, so we expect that this type of weak state appears with probabilities 2^{-480}, 2^{-608} and 2^{-768} for AEGIS-128, AEGIS-256 and AEGIS-128L, respectively.

7 The Performance of AEGIS

To process a 16-byte message block, AEGIS-128L, AEGIS-128 and AEGIS-256 use four, five and six AES round functions, respectively. In AEGIS, the critical path for processing a 16-byte message block is about one AES round. The computational cost of AEGIS is about half that of AES for each message block, thus the speed of AEGIS is about twice that of AES when they are implemented using table lookups. For implementations based on bit-slicing techniques (e.g. Käsper and Schwabe [13]), the difference is smaller as AEGIS-128 and AEGIS-256 allow for 5 and 6 parallel AES operations rather than 8; but the speed of AEGIS-128L is still about twice that of AES-128. AEGIS is very efficient when it is implemented using the AES new instructions (AES-NI) available on some x86 processors since 2010. With parallel AES round functions at each step, AEGIS can fully utilize the 3-stage pipeline in AES-NI in Intel Westmere processor, and can utilize most of the 8-stage pipeline in the AES-NI on the Intel Sandy Bridge processor. When implemented using AES-NI on the Sandy Bridge processor, the speed of AEGIS is about 8 times that of AES in CBC mode (encryption), and it is slightly faster than AES-CTR.

We implemented AEGIS in C code using AES-NI. We tested the speed on Intel Core i5-2540M 2.6GHz processor (Sandy Bridge) running 64-bit Ubuntu 11.04 and turning off the Turbo Boost. The compiler being used is gcc 4.5.2, and the options "-O3 -msse2 -maes -mavx" are used. In our test, associated data is not considered, and 128-bit tag is used. The test is performed by processing a message repeatedly and printing out the final message. To ensure that the tag generation is not removed in the compiler optimization process, we use the tag as IV for the next message. To ensure that the tag verification is not removed in the compiler optimization process, we count the number of failed verifications and print out the final result.

The performance is given in Table 2. For 4096-byte messages, the speed of AEGIS-128L is 0.48cpb; the speeds of AEGIS-128 and AEGIS-256 are around 0.7 cpb. According to Table 2, the performance of AEGIS is better than that of CCM, GCM and OCB3, ALE [5] and ASC-1 [11]. ALE and ASC-1 are two new authenticated encryption algorithms using AES instructions. In Table 2, the speed for multiple messages is not included since it is a common practice to compare the speeds for a single message. (For multiple long messages, the speeds of ALE and CCM are 1.2 and 3.1 cpb, respectively [5].) Note that the speeds given in Table 2 are for reference only since the ciphers are not evaluated under the same conditions.

In Table 2, AEGIS decryption-verification is slightly slower than encryption-authentication for two reasons: a ciphertext block needs to be decrypted first before it can be applied to update the state; and the verification process is

Table 2. The speed comparison (in cycles per byte) for different message length. A plus sign (+) indicates that the data are from the ALE designers and the performance is measured on the Intel i5-2400 microprocessor.

	64B	128B	256B	512B	1024B	4096B
AES-128-CTR$^+$	—	1.61	1.22	0.99	0.87	0.77
AES-128-CCM	7.26	6.31	5.65	5.19	5.17	5.05
AES-128-GCM$^+$	—	4.95	3.88	3.33	3.05	2.90
AES-128-OCB3$^+$	—	2.69	1.79	1.34	1.12	0.88
ALE$^+$	—	6.63	5.11	4.34	3.96	3.68
ASC-1$^+$	—	7.74	4.80	3.69	2.88	2.64
AEGIS-128L(EA[a])	3.68	2.05	1.23	0.83	0.63	0.48
AEGIS-128L(DV[b])	3.81	2.12	1.27	0.85	0.63	0.48
AEGIS-128(EA)	3.37	1.99	1.30	0.96	0.80	0.66
AEGIS-128(DV)	3.78	2.17	1.36	1.02	0.84	0.67
AEGIS-256(EA)	3.51	2.10	1.34	1.03	0.86	0.70
AEGIS-256(DV)	4.00	2.35	1.51	1.09	0.90	0.74

[a] EA: Encryption-Authentication
[b] DV: Decryption-Verification

slightly more expensive than tag generation. AEGIS-128 is only slightly slower than AEGIS-256 for long messages, although the computational cost of AEGIS-256 is about 20% more than that of AEGIS-128. The reason is that on the Sandy Bridge microprocessor, AES-NI is implemented with an eight-stage pipeline, and both AEGIS-128 and AEGIS-256 do not fully utilize the pipeline, so the performance of AEGIS-128 is close to that of AEGIS-256. On the Intel Westmere microprocessors with a 3-stage AES-NI, AEGIS-256 is about 20% slower than AEGIS-128.

8 Design Rationale

The goal of AEGIS is to achieve high performance and strong security. To achieve high performance, we use the AES round function which is now implemented on the latest Intel and AMD microprocessors as Intel AES New Instructions (AES-NI). AES-NI is very efficient for achieving diffusion and confusion on a modern microprocessor. In the design of AEGIS, we use several parallel AES round functions in each step so as to use most of the pipeline stages in AES instruction. AES instructions are implemented on Intel Westmere (06_25H, 06_2CH, 06_2FH) microprocessors with a three-stage pipeline (6 clock cycles), and are implemented on Intel Sandy Bridge (06_2AH) microprocessors with an eight-stage pipeline (8 clock cycles) [10]. Using several parallel AES round functions in AEGIS significantly improves its performance by utilizing the pipeline of AES-NI.

To achieve strong encryption security, we ensure that the IV difference is randomized at the initialization stage, and the state cannot be recovered from the ciphertext. There are at least 10 steps in the initialization of AEGIS, so we expect that the initialization of AEGIS is strong. To ensure that the state cannot be recovered from the ciphertext faster than brute force attack, we ensure that at least 20, 30 and 24 AES round functions are involved in the state recovery attack against AEGIS-128, AEGIS-256 and AEGIS-128L, respectively.

To achieve strong authentication security, we ensure that any difference being introduced into the state would result in a particular difference with sufficiently small probability so that it is difficult to launch a forgery attack. Our design is partly motivated by the design of Pelican MAC [6]. In Pelican MAC, a difference would pass through 4 AES round functions before meeting with another difference, so at least 25 active Sboxes are involved. The security proof against differential forgery attack is very simple for Pelican MAC (however, there is a birthday type attack against Pelican MAC due to its 128-bit size [28]). In AEGIS, the first difference in the state would pass through at least 4 AES round functions before being affected by another difference. In addition, when a difference passes through AES round functions, the differences are injected into at least four elements in the state, so it becomes more difficult to eliminate the state difference.

9 Conclusion

In this paper, we introduced a dedicated authenticated encryption algorithm AEGIS. AEGIS is fast for both short and long messages, and it is the fastest authenticated encryption algorithm on the microprocessors with the AES instruction set. We performed a security analysis of the encryption and authentication of AEGIS. Our analysis shows that the encryption and authentication of AEGIS are strong. We welcome the security analysis of this new authenticated encryption algorithm.

Finally we state that AEGIS is not patented and it is freely available for all applications.

Acknowledgements. We would like to thank the anonymous reviewers for their helpful comments, especially the idea of fully utilizing the 8-stage pipeline of AES-NI on the Sandy Bridge processor to achieve higher performance by increasing the state size. The first author has been funded by the NAP grant of the Nanyang Technological University. The second author has been funded in part by the Research Council KU Leuven (GOA TENSE) and the FWO Flanders.

References

1. M. Ågren, M. Hell, T. Johansson, W. Meier. Grain-128a: A New Version of Grain-128 with Optional Authentication. *International Journal of Wireless and Mobile Computing 2011*, Vol. 5, No. 1 pp. 48–59.

2. M. Bellare and C. Namprempre. Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm. *Advances in Cryptology – Asiacrypt 2000*, LNCS 1976, pp. 531–545.
3. A. Biryukov, The Design of a Stream Cipher LEX, *Selected Areas in Cryptography – SAC 2006*, LNCS 4356, pp. 67–75.
4. A. Biryukov. The Tweak for LEX-128, LEX-192, LEX-256. ECRYPT stream cipher project report 2006/037. Available at http://www.ecrypt.eu.org/stream.
5. A. Bogdanov, F. Mendel, F. Regazzoni, V. Rijmen, and E. Tischhauser. ALE: AES-Based Lightweight Authenticated Encryption. *Fast Software Encryption – FSE 2013*.
6. J. Daemen, V. Rijmen. The Pelican MAC Function. IACR Cryptology ePrint Archive 2005: 88 (2005).
7. O. Dunkelman, N. Keller. A New Attack on the LEX Stream Cipher. *Advances in Cryptology – Asiacrypt 2008*, LNCS 5350, pp. 539–556.
8. N. Ferguson, D. Whiting, B. Schneier, J. Kelsey, S. Lucks and T. Kohno. Helix, Fast Encryption and Authentication in a Single Cryptographic Primitive. *Fast Software Encryption – FSE 2003*, LNCS 2887, pp. 330–346.
9. V. Gligor and P. Donescu. Fast encryption and authentication: XCBC encryption and XECB authentication modes. *Fast Software Encryption – FSE 2001*, LNCS 2355, pp. 92–108.
10. Intel. Intel 64 and IA-32 Architectures Optimization Reference Manual. Available at http://www.intel.com/content/dam/doc/manual/64-ia-32-architectures-optimization-manual.pdf
11. G. Jakimoski and S. Khajuria. ASC-1: An Authenticated Encryption Stream Cipher. *Selected Area in Cryptography – SAC 2011*, LNCS 7118, pp. 356–372.
12. C. Jutla, Encryption modes with almost free message integrity. *Advances in Cryptology – EUROCRYPT 2001*, LNCS 2045, pp. 529–544.
13. E. Käsper and P. Schwabe. Faster and Timing-Attack Resistant AES-GCM. *Cryptographic Hardware and Embedded Systems – CHES 2009*, LNCS 5747, pp. 1–17.
14. J. Katz and M. Yung. Unforgeable encryption and adaptively secure modes of operation. *Fast Software Encryption–FSE 2000*, LNCS 1978, pp. 284–299.
15. T. Krovetz, P. Rogaway. The Software Performance of Authenticated-Encryption Modes. *Fast Software Encryption – FSE 2011*, LNCS 6733, pp. 306–327.
16. D. McGrew and J. Viega. The security and performance of the Galois/Counter Mode (GCM) of operation. *Progress in Cryptology – INDOCRYPT 2004*, LNCS 3348, pp. 343–355.
17. National Institute of Standards and Technology. Advanced Encryption Standard. FIPS 197.
18. National Institute of Standards and Technology. Recommendation for Block Cipher Modes of Operation. NIST special publication 800-38A, 2001 Edition.
19. National Institute of Standards and Technology. The Keyed-Hash Message Authentication Code (HMAC). FIPS PUB 198.
20. National Institute of Standards and Technology. Recommendations for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality. NIST special publication 800-38C, May 2004.
21. National Institute of Standards and Technology. Recommendations for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. NIST special publication 800-38D, November 2007.
22. National Institute of Standards and Technology. Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication. NIST special publication 800-38B.

23. B. Preneel, P. C. van Oorschot. On the Security of Iterated Message Authentication Codes. *IEEE Transactions on Information Theory 45(1)*, 188–199 (1999).
24. P. Rogaway, M. Bellare, and J. Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. ACM Trans. on Information and System Security, 6(3), pp. 365–403, 2003. Earlier version, with T. Krovetz, in CCS 2001.
25. P. Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. *Advances in Cryptology – ASIACRYPT 2004*, LNCS 3329, pp. 16–31.
26. D. Whiting, B. Schneier, S. Lucks and F. Muller. Phelix: Fast Encryption and Authentication in a Single Cryptographic Primitive. eSTREAM, ECRYPT Stream Cipher Project Report 2005/027.
27. H. Wu, B. Preneel. Differential-Linear Attacks Against the Stream Cipher Phelix. *Fast Software Encryption – FSE 2007*, LNCS 4593, pp. 87–100.
28. Z. Yuan, W. Wang, K. Jia, G. Xu, X. Wang. New Birthday Attacks on Some MACs Based on Block Ciphers. *Advances in Cryptology – CRYPTO 2009*, LNCS 5677, pp. 209–230.

A Test Vectors

The test vectors (in hexadecimal format) of AEGIS-128, AEGIS-256 and AEGIS-128L are given below.

A.1 Test vectors of AEGIS-128

```
associated data: 0 bits    plaintext: 128 bits
K128        = 00000000000000000000000000000000
IV128       = 00000000000000000000000000000000
plaintext   = 00000000000000000000000000000000
ciphertext  = 951b050fa72b1a2fc16d2e1f01b07d7e
tag         = a7d2a99773249542f422217ee888d5f1

associated data: 128 bits  plaintext: 128 bits
K128        = 00000000000000000000000000000000
IV128       = 00000000000000000000000000000000
assoc. data = 00000000000000000000000000000000
plaintext   = 00000000000000000000000000000000
ciphertext  = 10b0dee65a97d751205c128a992473a1
tag         = 46dcb9ee93c46cf13731d41b9646c131

associated data: 32 bits   plaintext: 128 bits
K128        = 00010000000000000000000000000000
IV128       = 00000200000000000000000000000000
assoc. data = 00010203
plaintext   = 00000000000000000000000000000000
ciphertext  = 2b78f5c1618da39afbb2920f5dae02b0
tag         = 74759cd0e19314650d6c635b563d80fd
```

```
associated data: 64 bits   plaintext: 256 bits
K128        = 10010000000000000000000000000000
IV128       = 10000200000000000000000000000000
assoc. data = 0001020304050607
plaintext   = 000102030405060708090a0b0c0d0e0f
              101112131415161718191a1b1c1d1e1f
ciphertext  = e08ec10685d63c7364eca78ff6e1a1dd
              fdfc15d5311a7f2988a0471a13973fd7
tag         = 27e84b6c4cc46cb6ece8f1f3e4aa0e78
```

A.2 Test vectors of AEGIS-256

```
associated data: 0 bits    plaintext: 128 bits
K128        = 00000000000000000000000000000000
              00000000000000000000000000000000
IV128       = 00000000000000000000000000000000
              00000000000000000000000000000000
plaintext   = 00000000000000000000000000000000
ciphertext  = b98f03a947807713d75a4fff9fc277a6
tag         = a008acb1d372d73932ec5e6df9aca70a

associated data: 128 bits  plaintext: 128 bits
K128        = 00000000000000000000000000000000
              00000000000000000000000000000000
IV128       = 00000000000000000000000000000000
              00000000000000000000000000000000
assoc. data = 00000000000000000000000000000000
plaintext   = 00000000000000000000000000000000
ciphertext  = b286705e6ccf368974ade9ff5550a4c5
tag         = 367f3f14897b31c6a66eb7b540eccc8b

associated data: 32 bits   plaintext: 128 bits
K128        = 00010000000000000000000000000000
              00000000000000000000000000000000
IV128       = 00000200000000000000000000000000
              00000000000000000000000000000000
assoc. data = 00010203
plaintext   = 00000000000000000000000000000000
ciphertext  = 1f452a22fc07f2471ab4345d7ab121b1
tag         = 0d80d9c73cd4b8b3422b66cdaa45ae8a

associated data: 64 bits   plaintext: 256 bits
K128        = 10010000000000000000000000000000
              00000000000000000000000000000000
IV128       = 10000200000000000000000000000000
```

```
               00000000000000000000000000000000
assoc. data = 0001020304050607
plaintext   = 000102030405060708090a0b0c0d0e0f
              101112131415161718191a1b1c1d1e1f
ciphertext  = f373079ed84b2709faee373584585d60
              accd191db310ef5d8b11833df9dec711
tag         = 787347bc96d3d0fdb33ddc8ee5ef4924
```

A.3 Test vectors of AEGIS-128L

```
associated data: 0 bits    plaintext: 128 bits
K128        = 00000000000000000000000000000000
              00000000000000000000000000000000
IV128       = 00000000000000000000000000000000
              00000000000000000000000000000000
plaintext   = 00000000000000000000000000000000
ciphertext  = 41de9000a7b5e40e2d68bb64d99ebb19
tag         = 8674521d074f983b2e830dd6f3edf4e5

associated data: 128 bits  plaintext: 128 bits
K128        = 00000000000000000000000000000000
              00000000000000000000000000000000
IV128       = 00000000000000000000000000000000
              00000000000000000000000000000000
assoc. data = 00000000000000000000000000000000
plaintext   = 00000000000000000000000000000000
ciphertext  = 29a0ce1f5dce8c404d56d00491668604
tag         = eb82ca639900a0699c859bfbf3020bfa

associated data: 32 bits   plaintext: 128 bits
K128        = 00010000000000000000000000000000
              00000000000000000000000000000000
IV128       = 00000200000000000000000000000000
              00000000000000000000000000000000
assoc. data = 00010203
plaintext   = 00000000000000000000000000000000
ciphertext  = 1c0f229f289844def2c1ef28bea0abf0
tag         = 86f3cc5e3a68f6e485960820be163808

associated data: 64 bits   plaintext: 256 bits
K128        = 10010000000000000000000000000000
              00000000000000000000000000000000
IV128       = 10000200000000000000000000000000
              00000000000000000000000000000000
assoc. data = 0001020304050607
plaintext   = 000102030405060708090a0b0c0d0e0f
```

```
               101112131415161718191a1b1c1d1e1f
ciphertext   = 79d94593d8c2119d7e8fd9b8fc77845c
               5c077a05b2528b6ac54b563aed8efe84
tag          = 4ed71cf3d6a3e568e8085110e92e8bfb
```