SAEAES

Designers/Submitters:
Yusuke Naito[1], Mitsuru Matsui[1],
Yasuyuki Sakai[1], Daisuke Suzuki[1],
Kazuo Sakiyama[2], Takeshi Sugawara[2]

1. Mitsubishi Electric Corporation, Japan
2. The University of Electro-Communications, Japan

Contact:
Naito.Yusuke@ce.MitsubishiElectric.co.jp

February 25, 2019

1 Introduction

We propose SAEAES, a family of AES-based AEAD (Authenticated Encryption with Associated Data) schemes suitable for lightweight applications. SAEAES is an instantiation of the block cipher-based mode of operation SAEB [13] with the standard block cipher AES [17]. SAEB stands for **S**mall (Simple, Slim, Sponge-based) **AE**AD from **B**lock cipher). SAEAES is named by replacing the B for block cipher with AES.

SAEB: a lightweight AEAD mode of operation for block ciphers published at CHES2018 [13]. It is designed with the five requirements: (1) minimum state size, (2) inverse free, (3) XOR only, (4) online, and (5) efficient handling of static associated data. SAEB realizes compact implementations with respect to RAM/resister size and ROM/circuit size.

AES: the block cipher standardized by NIST as the Federal Information Processing Standard Publications (FIPS Pub) 197 [17]. Its security and implementation have been extensively studied. Many computational platforms offer AES accelerators in the form of a special instruction or a coprocessor. Current standard AEAD schemes such as AES-CCM [16] and AES-GCM [15] are also based on AES. Therefore, the users of these schemes can easily migrate to SAEAES.

The document is organized as follows. The specification of SAEAES is described in Section 2. The security claims of SAEAES are given in Section 3. The design rationale of SAEAES is shown in Section 4. The implementation results are summarized in Section 5. We give a brief comment on the third party analysis in Section 6.

2 Specification

We describe the SAEB mode of operation followed by the specification of the SAEAES family.

2.1 SAEB

Basic Notations. In this section, the following notations are used.

- λ: an empty string.
- 0^i: an i-bit string of i zeros for an integer $i \geq 0$, and $0^0 := \lambda$.
- $\mathbf{i}_j$: a j-bit representation of i for integers $j, i > 0$, e.g, $\mathbf{1}_i = 0^{i-1}1$, $\mathbf{2}_i = 0^{i-2}10$, and $\mathbf{3}_i = 0^{i-2}11$
- $|x|$: the bit length of a bit string x.
- $\mathsf{msb}_l(x)$: the most significant l-bit string of a bit string x for an integer $l \geq 0$ such that $l \leq |x|$.
- $\mathsf{lsb}_l(x)$: the least significant l-bit string of a bit string x for an integer $l \geq 0$ such that $l \leq |x|$.
- $\{0,1\}^*$: the set of all bit strings.
- $\{0,1\}^i$: the set of i-bit strings for an integer $i \geq 0$ and $\{0,1\}^0 := \{\lambda\}$.
- $\{0,1\}^{\leq i} := \{0,1\}^0 \cup \{0,1\}^1 \cup \cdots \cup \{0,1\}^i$ the set of bit strings whose bit lengths are equal to or less than i for an integer $i \geq 0$.
- $(x_1, \ldots, x_l) \xleftarrow{r} x$: a bit string x is partitioned into l blocks $x_1, \ldots, x_l$ for an integer $r \geq 0$, such that $x = x_1 \| \cdots \| x_l$, $|x_i| = r$ for $i = 1, \ldots, l-1$, and if $x \neq \lambda$ and $|x| < lr$ then $1 \leq |x_l| < r$; if $x \neq \lambda$ and $|x| = lr$ then $|x_l| = r$; if $x = \lambda$ then $l = 1$ and $x_1 = \lambda$.

Notations for the Underlying Block Cipher. We use the following notations for the underlying block cipher of SAEB.

- k: a positive integer and the key length of the underlying block cipher.
- n: a positive integer greater than 2 and the block length of the underlying block cipher.
- $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$: the underlying block cipher of SAEB.
- $K \in \{0,1\}^k$: a block cipher key.
- $E_K : \{0,1\}^n \to \{0,1\}^n$: the underlying block cipher having a key $K \in \{0,1\}^k$.

Internal Parameters. SAEB is a nonce-based AEAD mode of operation. The internal parameters of SAEB are defined as follows.

- r_1: a positive integer less than $n-1$ and the bit length of an associated data block.
- r_2: a positive integer less than $n-1$ and the bit length of a nonce.
- r: a positive integer less than $n-1$ and the bit length of a plaintext/ciphertext block.
- τ: a positive integer equal to or less than n and the bit length of an authentication tag.
- $c_1 := n - r_1$.
- $c_2 := n - r_2$.
- $c := n - r$.
- $\mathcal{K} := \{0,1\}^k$: the set of keys of SAEB.
- $\mathcal{N} := \{0,1\}^{r_2}$: the set of nonces of SAEB.
- $\mathcal{P} := \{0,1\}^*$: the set of plaintexts of SAEB.
- $\mathcal{C} := \{0,1\}^*$: the set of ciphertexts of SAEB.
- $\mathcal{AD} := \{0,1\}^*$: the set of associated data of SAEB.
- $\mathcal{T} := \{0,1\}^\tau$: the set of tags of SAEB.

One-Zero Padding. SAEB uses a one-zero padding function. For an integer $i > 0$ and a bit string x such that $|x| < i$, the one-zero padding function $\mathsf{ozp}_i : \{0,1\}^{\leq i-1} \to \{0,1\}^i$ is defined as $\mathsf{ozp}_i(x) = x \| 10^{i-1-|x|}$.

Specification of SAEB. The encryption algorithm of SAEB using a keyed block cipher E_K, denoted by $\mathtt{SAEB.Enc}[E_K]$, takes the following three inputs

- an r_2-bit nonce $N \in \mathcal{N}$,
- variable-length associated data $A \in \mathcal{AD}$,
- a variable-length plaintext $P \in \mathcal{P}$,

and returns a pair of

- a ciphertext $C \in \{0,1\}^{|P|}$,
- a τ-bit authentication tag $T \in \mathcal{T}$.

The decryption algorithm of SAEB using a keyed block cipher E_K, denoted by $\mathtt{SAEB.Dec}[E_K]$, takes the following three inputs

- an r_2-bit nonce $N \in \mathcal{N}$,
- variable-length associated data $A \in \mathcal{AD}$,
- a pair of a variable-length ciphertext $C \in \mathcal{C}$ and a τ-bit authentication tag $T \in \mathcal{T}$,

and returns either

- the invalid symbol $\perp \notin \mathcal{P}$ or
- a plaintext $P \in \{0,1\}^{|C|}$.

Algorithm 1 shows the encryption and decryption algorithms of SAEB. In these algorithms, the subroutine Hash is used to process associated data and a nonce. Fig. 1 shows Hash for empty associated data $A = \lambda$ (left) and non-empty associated data $A \neq \lambda$ (right). In the encryption routine SAEB.Enc, the core algorithm Core.Enc processes a plaintext P and generates an authentication tag T. Fig. 2 shows Core.Enc for an empty plaintext $P = \lambda$ (left) and a non-empty plaintext $P \neq \lambda$ (right). In the decryption routine SAEB.Dec, the core algorithm Core.Dec processes a ciphertext C and generates an authentication tag T'. Fig. 3 shows Core.Dec for an empty ciphertext $C = \lambda$ (left) and a non-empty ciphertext $C \neq \lambda$ (right).

Algorithm 1 SAEB

▶ Encryption $\texttt{SAEB.Enc}[E_K](N, A, P)$

1: $\mathsf{iv} \leftarrow \texttt{Hash}[E_K](N, A)$
2: $(C, T) \leftarrow \texttt{Core.Enc}[E_K](\mathsf{iv}, P)$
3: **return** (C, T)

▶ Decryption $\texttt{SAEB.Dec}[E_K](N, A, (C, T))$

1: $\mathsf{iv} \leftarrow \texttt{Hash}[E_K](N, A)$
2: $(P, T') \leftarrow \texttt{Core.Dec}[E_K](\mathsf{iv}, C)$
3: **if** $T = T'$ **then return** P
4: **if** $T \neq T'$ **then return** $\perp$

▷ Subroutine $\texttt{Hash}[E_K](N, A)$

1: $\mathsf{sa} \leftarrow 0^n$; $A_1, \ldots, A_a \xleftarrow{r_1} A$
2: **for** $i = 1$ to $a - 1$ **do** $\mathsf{msb}_{r_1}(\mathsf{sa}) \leftarrow \mathsf{msb}_{r_1}(\mathsf{sa}) \oplus A_i$; $\mathsf{sa} \leftarrow E_K(\mathsf{sa})$
3: **if** $|A_a| = r_1$ **then** $\mathsf{sa} \leftarrow \mathsf{sa} \oplus (A_a \| \mathbf{1}_{c_1})$
4: **if** $|A_a| < r_1$ **then** $\mathsf{sa} \leftarrow \mathsf{sa} \oplus (\mathsf{ozp}_{r_1}(A_a) \| \mathbf{2}_{c_1})$
5: $\mathsf{sa} \leftarrow E_K(\mathsf{sa})$; $\mathsf{iv} \leftarrow \mathsf{sa} \oplus (N \| \mathbf{3}_{c_2})$
6: **return** iv

▷ Subroutine $\texttt{Core.Enc}[E_K](\mathsf{iv}, P)$

1: $\mathsf{sm} \leftarrow E_K(\mathsf{iv})$; $P_1, \ldots, P_p \xleftarrow{r} P$
2: **for** $i = 1$ to $p - 1$ **do** $\mathsf{msb}_r(\mathsf{sm}) \leftarrow \mathsf{msb}_r(\mathsf{sm}) \oplus P_i$; $C_i \leftarrow \mathsf{msb}_r(\mathsf{sm})$; $\mathsf{sm} \leftarrow E_K(\mathsf{sm})$
3: **if** $|P_p| = r$ **then** $\mathsf{sm} \leftarrow \mathsf{sm} \oplus (P_p \| \mathbf{1}_c)$; $C_p \leftarrow \mathsf{msb}_r(\mathsf{sm})$
4: **if** $|P_p| < r$ **then** $\mathsf{sm} \leftarrow \mathsf{sm} \oplus (\mathsf{ozp}_r(P_p) \| \mathbf{2}_c)$; $C_p \leftarrow \mathsf{msb}_{|P_p|}(\mathsf{sm})$
5: $\mathsf{sm} \leftarrow E_K(\mathsf{sm})$; $T \leftarrow \mathsf{msb}_\tau(\mathsf{sm})$
6: **return** $(C_1 \| C_2 \| \cdots \| C_p, T)$

▷ Subroutine $\texttt{Core.Dec}[E_K](\mathsf{iv}, C)$

1: $\mathsf{sm} \leftarrow E_K(\mathsf{iv})$; $C_1, \ldots, C_p \xleftarrow{r} C$
2: **for** $i = 1$ to $p - 1$ **do** $P_i \leftarrow \mathsf{msb}_r(\mathsf{sm}) \oplus C_i$; $\mathsf{msb}_r(\mathsf{sm}) \leftarrow P_i \oplus \mathsf{msb}_r(\mathsf{sm})$; $\mathsf{sm} \leftarrow E_K(\mathsf{sm})$
3: **if** $|C_p| = r$ **then** $P_p \leftarrow \mathsf{msb}_r(\mathsf{sm}) \oplus C_p$; $\mathsf{sm} \leftarrow \mathsf{sm} \oplus (P_p \| \mathbf{1}_c)$
4: **if** $|C_p| < r$ **then** $P_p \leftarrow \mathsf{msb}_{|C_p|}(\mathsf{sm}) \oplus C_p$; $\mathsf{sm} \leftarrow \mathsf{sm} \oplus (\mathsf{ozp}_r(P_p) \| \mathbf{2}_c)$
5: $\mathsf{sm} \leftarrow E_K(\mathsf{sm})$; $T' \leftarrow \mathsf{msb}_\tau(\mathsf{sm})$
6: **return** $(P_1 \| P_2 \| \cdots \| P_p, T')$

2.2 Specification of SAEAES

Interface. The SAEAES encryption algorithm receives four byte-string inputs and returns a byte-string output. The four inputs are (i) a variable-length plaintext, (ii) variable-length associated data, (iii) a fixed-length nonce, and (iv) a fixed-length key. The single output is the concatenation of a variable-length ciphertext and an authentication tag.

The SAEAES decryption algorithm receives four byte-string inputs and returns a byte-string output. The four inputs are (i) the concatenation of a variable-length ciphertext and an authentication tag, (ii) variable-length associated data, (iii) a fixed-length nonce, and (iv) a fixed-length key. If the verification is successful, the output is a variable-length plaintext. Otherwise, the output is an invalid symbol.

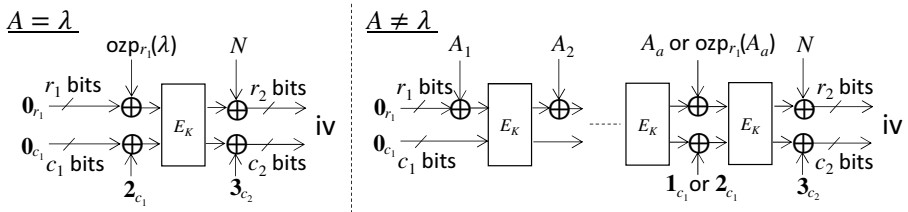

Fig. 1. Hash.

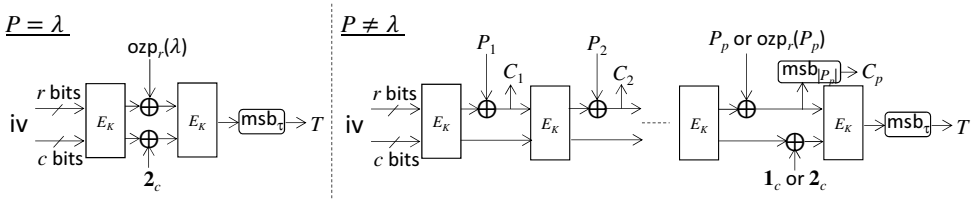

Fig. 2. Core.Enc.

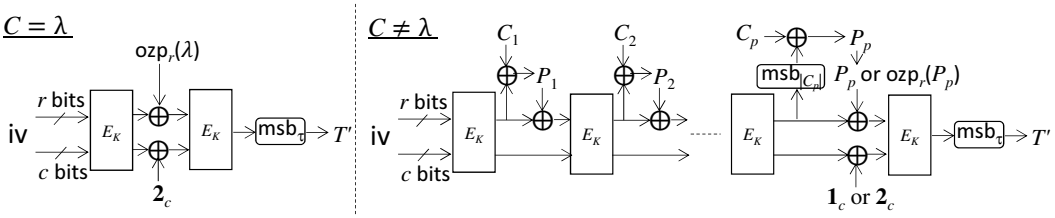

Fig. 3. Core.Dec.

Parameter Candidates. The following parameters are fixed in the SAEAES family:

- Block length of the underlying block cipher n : 128 bits (16 bytes).
- Nonce length r_2 : 120 bits (15 bytes).
- Plaintext/ciphertext block length r : 64 bits (8 bytes).

The following parameters vary depending on a member of the SAEAES family:

- Key length k : 128 bits (16 bytes), 192 bits (24 bytes), 256 bits (32 bytes).
- associated data block length r_1 : 64 bits (8 bytes), 120 bits (15 bytes).
- Tag length τ : 64 bits (8 bytes), 128 bits (16 bytes).

Block Ciphers. SAEAES uses the standard block cipher AES of FIPS Pub 197 [17]. AES can process a 128-bit data block, using a key of length 128, 192, or 256 bits.

- For the members of SAEAES with $k = 128$, the 128-bit key version of AES denoted by AES-128 is used.

Table 1. Parameters

Identifier	k	n	r_2	r_1	r	τ
SAEAES128_64_64	128	128	120	64	64	64
SAEAES128_64_128	128	128	120	64	64	128
SAEAES128_120_64	128	128	120	120	64	64
SAEAES128_120_128	128	128	120	120	64	128
SAEAES192_64_64	192	128	120	64	64	64
SAEAES192_64_128	192	128	120	64	64	128
SAEAES192_120_128	192	128	120	120	64	128
SAEAES256_64_64	256	128	120	64	64	64
SAEAES256_64_128	256	128	120	64	64	128
SAEAES256_120_128	256	128	120	120	64	128

- For the members of SAEAES with $k = 192$, the 192-bit key version of AES denoted by AES-192 is used.
- For the members of SAEAES with $k = 256$, the 256-bit key version of AES denoted by AES-256 is used.

SAEAES uses the Advanced Encryption Standard (AES) algorithm. We decided not to include the specification of AES in this document for the sake of avoiding ambiguity. We refer to FIPS Pub 197 [17] for the complete specification of AES.

Members of SAEAES. Table 1 summarizes the 10 members of SAEAES and their parameters. A member is determined by a triplet (k, r_1, τ) and the member with (k, r_1, τ) is identified by SAEAES$k_r_1_\tau$. For each of the 10 members, the lengths of associated data and plaintexts per key should be at least 0 byte and at most 2^{64}-1 bytes.

Primary Member. SAEAES128_64_128 is the primary member of SAEAES.

3 Security

3.1 Security Claims

The security of SAEAES is ensured with respect to the indistinguishability between SAEAES and an ideal nonce-based AEAD scheme, under a single-key and nonce-respecting setting. Adversary's queries to SAEAES or the ideal AEAD scheme are called online queries, and queries to the encryption resp. decryption oracle is called encryption resp. decryption queries. The decryption queries mean forgery attempts by an adversary. Hereafter, the time complexity of an adversary is called offline complexity. The security claims of the members of SAEAES are summarized as followings.

- Attacks on the members of SAEAES with 128-bit key, SAEAES128_64_64, SAEAES128_64_128, SAEAES128_120_64, SAEAES128_120_128, require
  - at least 2^{112} offline complexity,
  - at least 2^{62} keyed block cipher calls in total by all encryption queries,
  - at least 2^{58} keyed block cipher calls in total by all decryption queries, or
  - at least 2^{58} keyed block cipher calls in total in the Hash procedures by all encryption queries.
- Attacks on the members of SAEAES with 192-bit key, SAEAES192_64_64, SAEAES192_64_128, SAEAES192_120_128, require
  - at least 2^{168} offline complexity,
  - at least 2^{62} keyed block cipher calls in total by all encryption queries,
  - at least 2^{58} keyed block cipher calls in total by all decryption queries, or
  - at least 2^{58} keyed block cipher calls in total in the Hash procedures by all encryption queries.
- Attacks on the members of SAEAES with 256-bit key, SAEAES256_64_64, SAEAES256_64_128, SAEAES256_120_128, require
  - at least 2^{224} offline complexity,
  - at least 2^{62} keyed block cipher calls in total by all encryption queries,
  - at least 2^{58} keyed block cipher calls in total by all decryption queries, or
  - at least 2^{58} keyed block cipher calls in total in the Hash procedures by all encryption queries.

The above claims are ensured by combining the following two steps.

Security Claims of AES. The first step considers the pseudo-random-permutation (PRP) security of AES-128, AES-192 and AES-256. The definition of PRP-security is given in Subsection 3.2. We claim the following security levels of these block ciphers.

- Attacks on AES-128 require at least 2^{112} offline complexity or at least 2^{64} keyed block cipher calls by online queries.
- Attacks on AES-192 require at least 2^{168} offline complexity or at least 2^{64} keyed block cipher calls by online queries.
- Attacks on AES-256 require at least 2^{224} offline complexity or at least 2^{64} keyed block cipher calls by online queries.

These claims are based on the existing cryptanalyses on AES, and the cryptanalyses are summarized in Subsection 3.3.

Security Claims of SAEAES with Secure PRP. The second step considers the indistinguishability between SAEAES and an ideal nonce-based AEAD scheme, where the underlying keyed block cipher is assumed to be a secure PRP (more precisely, the PRP-security advantage of the keyed block cipher is negligible). The security definition (nAE-security) is given in Subsection 3.2. We claim the following security levels of SAEAES with a secure PRP.

- Attacks on SAEAES with a secure PRP require
  - at least 2^{62} keyed block cipher calls in total by all encryption queries,
  - at least 2^{58} keyed block cipher calls in total by all decryption queries, or
  - at least 2^{58} keyed block cipher calls in total in the Hash procedures by all encryption queries.

The claim is based on the security of SAEB, and the security bound is given in Subsection 3.4.

3.2 Security Definitions

Notations. In this section, the notations given in Subsection 2.1 and the following ones are used. For a finite set $\mathcal{X}$, $x \twoheadleftarrow \mathcal{X}$ means that an element is drawn uniformly at random from $\mathcal{X}$ and is assigned to x. Let $\mathsf{Perm}(\mathcal{B})$ be the set of all permutations over a non-empty set $\mathcal{B}$. A random permutation in $\mathsf{Perm}(\mathcal{B})$ is defined as $\pi \twoheadleftarrow \mathsf{Perm}(\mathcal{B})$. An output of an adversary $\mathbf{A}$ with oracle access to $\mathcal{O}$ is denoted by $\mathbf{A}^{\mathcal{O}}$. Let $E = \texttt{AES-128}$ if $k = 128$, $E = \texttt{AES-192}$ if $k = 192$ and $E = \texttt{AES-256}$ if $k = 256$. Let $\Pi = \texttt{SAEAES}k\_r_1\_\tau$. Let $\Pi.\texttt{Enc}[E_K]$ be the encryption algorithm of Π using E_K, and $\Pi.\texttt{Dec}[E_K]$ be the decryption algorithm of Π using E_K.

Security Definition of Block Cipher. The security of SAEAES is ensured as long as E_K is a secure pseudo-random permutation (PRP). The advantage function of a PRP-adversary $\mathbf{A}$ that outputs a bit is defined as

$$\mathbf{Adv}^{\mathsf{prp}}_{E}(\mathbf{A}) = \Pr[K \twoheadleftarrow \{0,1\}^k; \mathbf{A}^{E_K} = 1] - \Pr[\pi \twoheadleftarrow \mathsf{Perm}(\mathcal{B}); \mathbf{A}^{\pi} = 1] \ ,$$

where the probabilities are taken over $\mathbf{A}$, K and π.

Security Definition of AEAD. The indistinguishability between $(\Pi.\texttt{Enc}[E_K], \Pi.\texttt{Dec}[E_K])$ and an ideal AEAD scheme $(\$, \perp)$, used in [14, 19], is considered where $\$$ is a random-bits oracle that has the same interface as $\Pi.\texttt{Enc}[E_K]$ and for query (N, A, P) returns a random bit string of length $|\Pi.\texttt{Enc}[E_K](N, A, P)|$; $\perp$ is an oracle that returns the reject symbol $\perp$ for any query. The security is called nAE-security. Formally, we consider an adversary $\mathbf{A}$ that first interacts with either $(\Pi.\texttt{Enc}[E_K], \Pi.\texttt{Dec}[E_K])$ or $(\$, \perp)$, and then returns a decision bit $b \in \{0,1\}$. The nAE-advantage function of $\mathbf{A}$ is defined as

$$\mathbf{Adv}^{\mathsf{nAE}}_{\Pi}(\mathbf{A}) = \Pr[K \twoheadleftarrow \mathcal{K}; \mathbf{A}^{\Pi.\texttt{Enc}[E_K], \Pi.\texttt{Dec}[E_K]} = 1] - \Pr[\mathbf{A}^{\$,\perp} = 1] \ .$$

We demand that $\mathbf{A}$ is nonce-respecting (all nonces by encryption queries are distinct), that $\mathbf{A}$ never asks a trivial decryption query $(N, A, (C, T))$ such that there is a prior encryption query (N, A, P) with $(C, T) = \Pi.\texttt{Enc}[E_K](N, A, P)$, and that $\mathbf{A}$ never repeats a query.

Table 2. Summary of cryptanalysis on `AES` in the single-key setting. "Time" shows offline complexity, "Data" shows the number of keyed block cipher calls, and "Memory" shows memory sizes in blocks. The second attack on the 9-round `AES-192` (3-11) in [10] considers `AES-192` from rounds 3 to 11.

Identifier	Rounds	Time	Data	Memory	Technique	Ref.
`AES-128`	7	2^{97}	2^{99}	2^{98}	MitM	[9]
	10	$2^{126.59}$	2	2^{60}	Biclique	[7]
	10	$2^{125.99}$	2^{56}	2^{60}	Biclique	[20]
`AES-192`	8	2^{172}	2^{107}	2^{96}	MitM	[9]
	9	$2^{186.5}$	2^{121}	$2^{177.5}$	MitM	[10]
	9 (3-11)	$2^{182.5}$	2^{117}	$2^{165.5}$	MitM	[10]
	12	$2^{190.83}$	2	2^{60}	Biclique	[7]
	12	$2^{189.76}$	2^{48}	2^{60}	Biclique	[20]
`AES-256`	9	2^{203}	2^{120}	2^{203}	MitM	[9]
	10	2^{253}	2^{111}	$2^{211.2}$	MitM	[11]
	14	$2^{254.94}$	3	2^{60}	Biclique	[7]
	14	$2^{254.28}$	2^{40}	2^{60}	Biclique	[20]

3.3 Security of `AES`

We summarize the state-of-the-art cryptanalyses of `AES` that `SAEAES` relies on. So far, no practical single-key attack against AES is known.

So far, the best single-key attack of full-round `AES` is the biclique attack. It was proposed by Bogdanov et al. [8] and then improved by Bogdanov et al. [7] and Tao and Wu [20]. It has a slight advantage over brute-force attack as summarized in Table 2. However, as described by the authors of the original paper [8], it does not threaten the practical use of `AES` in any way. Besides, for reduced-round variants of `AES`, there are improved attacks based on meet-in-the-middle (MitM) attack as summarized in Table 2.

In summary, there is no attack in the single-key setting

- that breaks the PRP-security of `AES-128` with less than 2^{112} offline complexity or 2^{64} keyed block cipher calls,
- that breaks the PRP-security of `AES-192` with less than 2^{168} offline complexity or 2^{64} keyed block cipher calls, or
- that breaks the PRP-security of `AES-256` with less than 2^{224} offline complexity or 2^{64} keyed block cipher calls.

Remark 1. We also note that there are efficient related-key attacks on full-round `AES-192` and `AES-256` [6, 5]. However, it does not affect the security of `SAEAES` because we claim the single-key security only.

3.4 Security of `SAEAES` with Secure PRP

The nAE-security bound of `SAEB` is given below. The proof is given in [13].

Theorem 1. *Let* $\mathbf{A}$ *be an nAE-adversary that makes* $q_{\mathcal{E}}$ *encryption queries and* $q_{\mathcal{D}}$ *decryption queries, and runs in time* t*. Let* $\sigma_{\mathcal{E}}$ *and* $\sigma_{\mathcal{D}}$ *be the total numbers of block cipher calls by encryption and decryption queries, respectively, and* $\sigma := \sigma_{\mathcal{E}} + \sigma_{\mathcal{D}}$ *the total number of block cipher calls by all queries. Let* σ_A *be the total number of block cipher calls*

in `Hash` *in* `SAEB.Enc` *(only encryption queries). For any positive integer* ρ*, there exists a PRP-adversary* $\mathbf{B}$ *making at most* σ *queries and running in time* $t + O(\sigma)$ *such that*

$$\mathbf{Adv}^{\mathsf{nAE}}_{\mathtt{SAEB}}(\mathbf{A}) \leq \frac{2\sigma^2}{2^n} + \frac{(\rho-1)(\sigma_A+\sigma_{\mathcal{D}})}{2^c} + 2^r\left(\frac{e\sigma_{\mathcal{E}}}{\rho 2^r}\right)^{\rho} + \frac{q_{\mathcal{D}}}{2^\tau} + \mathbf{Adv}^{\mathsf{prp}}_E(\mathbf{B}) \ .$$

Putting the parameters of the members of `SAEAES`, i.e., $r = c = 64$ (and $n = 128$) and putting $\rho = 16$,

$$\begin{aligned}\mathbf{Adv}^{\mathsf{nAE}}_{\mathtt{SAEB}}(\mathbf{A}) &\leq \frac{2\sigma^2}{2^{128}} + \frac{15\cdot(\sigma_A+\sigma_{\mathcal{D}})}{2^{64}} + 2^{64}\cdot\left(\frac{e\sigma_{\mathcal{E}}}{16\cdot 2^{64}}\right)^{16} + \frac{q_{\mathcal{D}}}{2^\tau} + \mathbf{Adv}^{\mathsf{prp}}_E(\mathbf{B}) \\ &\leq \frac{2\sigma^2}{2^{128}} + \frac{15\cdot(\sigma_A+\sigma_{\mathcal{D}})}{2^{64}} + \left(\frac{e\sigma_{\mathcal{E}}}{2^{64}}\right)^{16} + \frac{q_{\mathcal{D}}}{2^\tau} + \mathbf{Adv}^{\mathsf{prp}}_E(\mathbf{B}) \ .\end{aligned}$$

Hence, assuming the PRP-advantage $\mathbf{Adv}^{\mathsf{prp}}_E(\mathbf{B})$ is negligible compared with other terms in the bound, the bound is less than $1/2$ as long as $\sigma_{\mathcal{E}} \leq 2^{62}$, $\sigma_{\mathcal{D}} \leq 2^{58}$, and $\sigma_A \leq 2^{58}$.

3.5 Security of `SAEAES` Under Nonce Misuse

The nAE-security of `SAEAES` is broken under a nonce-misuse setting, where the same nonce is repeated for some encryption queries in the single-key setting.

Theorem 2. *For each of members of* `SAEAES`*, denoted by* Π*, there exists a nonce-misuse adversary* $\mathbf{A}$ *that makes two distinct encryption queries and runs in time* $O(1)$ *such that*

$$\mathbf{Adv}^{\mathsf{nAE}}_{\Pi}(\mathbf{A}) \geq 1 - \frac{1}{2^r} \ .$$

Proof. A nonce-misuse adversary $\mathbf{A}$ (the same nonce is repeated) is defined below.

- Make an encryption query $(N^{(1)}, A^{(1)}, P^{(1)}) = (0^{r_2}, \lambda, 0^r)$ and receive the ciphertext $C^{(1)}$ and the tag $T^{(1)}$.
- Make an encryption query $(N^{(2)}, A^{(2)}, P^{(2)}) = (0^{r_2}, \lambda, 1^r)$ and receive the ciphertext $C^{(2)}$ and the tag $T^{(2)}$.
- If $C^{(1)} \oplus C^{(2)} = 1^r$ then return 1.
- Return 0.

In the world with `SAEAES`, the probability that $\mathbf{A}$ returns 1 is 1. On the other hand, in the world with an ideal nonce-based AEAD scheme, the first r-bit ciphertext blocks of $C^{(1)}$ and $C^{(2)}$ are chosen independently and uniformly at random from $\{0,1\}^r$, thus the probability that $\mathbf{A}$ returns 1 (i.e., $C^{(1)} \oplus C^{(2)} = 1^r$) is $1/2^r$. Hence, the lower bound in Theorem 2 is obtained.

□

4 Design Rationale

4.1 Design Goal of SAEB

SAEB is designed as a lightweight block cipher-based AEAD scheme that can be implemented with limited computational resources both in software and in hardware. We prioritized the following four desirable properties for lightweight AEAD schemes.

1. **Minimum State Size:** The state size is equal to the block size n, which is the minimal amount of memory required to keep storing the internal sate during the AEAD computation. The n-bit state is minimal in the sense that at least n-bit state is necessary to maintain the birthday security.
2. **Inverse Free:** A block cipher decryption is not needed. This property obviously contributes to lightweight implementation, in particular when the structure of decryption of a block cipher is significantly different that of encryption, such as AES.
3. **XOR Only:** The entire AEAD scheme consists of a block cipher component and XOR operations only. Equivalently, its mode-of-operation part consists of XOR operations only. This property also makes small and fast implementation possible. Note that AES-GCM requires an additional component GHASH, which cannot be implemented with XOR operations only,
4. **Online:** An input data stream is sequentially processed only once without necessity of storing the entire stream for second read. An offline algorithm must prepare a buffer memory that can accommodate a data stream of the maximal possible length, which is costly and not suitable for embedded applications.

On top of the above properties, the fifth property regarding associated data is added:

5. **Efficient Handling of Static Associated Data:** If the same associated data is used repeatedly (i.e. static associated data), the second and later times can be faster than the first time due to the precomputation phase thereby reducing computational cost [18]. AES-GCM and many other block cipher based AEAD schemes satisfy this property.

SAEB is designed to satisfy all the five properties.

4.2 Design of SAEB

SAEB follows the sponge-based design methodology [2, 3, 4] except that it is based on a block cipher, not a permutation. SAEB can be thought as a cascaded n-bit block cipher. The state size of SAEB is n bits, and a block cipher decryption is not used. Therefore, the properties 1 and 2 are satisfied. All input data (either of associated data, nonce, plaintext and ciphertext) are absorbed into the internal state using XOR operations before/after an internal block cipher. Hence, the properties 3 and 4 are satisfied. Since associated data is absorbed earlier than the nonce, the internal state value right after processing associated data (and before processing nonce) is independent of the nonce value, and thus the property 5 is satisfied.

4.3 The Choice of Block Cipher

We decided to use AES because it is standardized, is extensively studied in the research community, and is widely used in industry.

5 Implementations

5.1 Embedded Software Implementations

SAEAES has been carefully designed to be particularly suited to embedded applications with resource constrained environments. Each of its hash, encryption and decryption functions has almost the same structure with common components, which makes possible software implementation with a small ROM size, and moreover only one 16-byte buffer is required to keep storing its internal state during the entire computation, which significantly contributes to RAM size reduction.

SAEAES calls an AES block cipher component internally without necessity of its inversion function (i.e. inverse free), and the overhead of its 'mode-of-operation part', which consists of a small number of XOR operations only, is extremely small. Hence it is easy to estimate the encryption speed of SAEAES if you have an existing AES implementation. For instance, the expected number of cycles required for encrypting $8a$-byte associated data and $8m$-byte message is simply $(1+a+m)e$, where e denotes the number of cycles required for one block AES encryption. Also note that since nonce N is embedded after the hash function, the number of cycles can be reduced to $(1+m)e$, independent of the value of a, under the scenario that a fixed associated data is used repeatedly with the same key.

Tables 3 and 4 present our implementation results of SAEAES128_64_128 on Renesas RL78 16-bit microcontroller, which has been used widely in industrial applications such as vehicle systems. RL78 is a typical accumulator-based CISC processor with eight 8-bit general registers a,x,b,c,d,e,h,l, which can be also used as four 16-bit general register pairs ax,bc,de,hl. Our AEAD encryption and decryption codes are written in assembly language as a subroutine callable from C language and parameters are passed as external variables in order to minimize ROM/RAM size. We designed two types of implementation: one for aiming at fast speed and the other for small memory size. All of our codes have resistance against timing attacks since they run exactly in the same number of cycles as long as the length of data is the same.

Table 3 shows the number of ROM/RAM bytes of our codes that include both AEAD encryption and decryption. For comparison, our implementation results of AES-GCM and AES are also listed, where the AES code, which has adopted the on-the-fly key scheduling algorithm for minimizing RAM usage of subkey, is used in SAEAES and AES-GCM internally. The RAM size includes stack consumption but not parameters. As shown in this table, the memory size of the mode of operation of SAEAES is much smaller than that of the internal AES routine. In particular, it should be noted that the ROM size of the mode-of-operation part of our small code is only 180 bytes.

Table 4 shows encryption speed of our codes of SAEAES and AES-GCM with 8-byte associated data and n-byte message for $n = 16, 128, 1024$ and ∞. Note that the internal AES function is called twice in the hash function of SAEAES when the length l of associated data is $0 \le l \le 8$. Also the decryption speed of SAEAES is almost the same as its encryption speed. The bottom row presents performance ratio of $(1 + a + m)e$ shown above (a=1 in this case), which demonstrates that the performance overhead of the mode of operation of SAEAES is negligible.

5.2 Hardware Implementations

Another important target of lightweight cryptography is hardware implementations in *application specific integrated circuit* (ASIC) and *field-programmable gate array* (FPGA).

Table 3. Implementation Results of SAEAES128_64_128 on RL78 (ROM/RAM bytes)

	Fast		Small	
	ROM	RAM	ROM	RAM
SAEAES	1449	46	581	74
AES-GCM	1533	158	983	178
AES	930	26	401	48

Table 4. Implementation Results of SAEAES128_64_128 on RL78 (cycles/byte)

	Fast				Small			
Message Size	16bytes	128bytes	1Kbytes	Maximum	16bytes	128bytes	1Kbytes	Maximum
SAEAES	915	508	457	450	2240	1250	1126	1109
AES-GCM	3811	1589	1312	1272	9371	3906	3223	3125
AES	222				544			
SAEAES/$(2+m)$AES	1.028	1.015	1.012	1.011	1.029	1.021	1.019	1.019

We describe the lightweight hardware architecture appeared in the original paper [13] (Fig. 4).

The implementation is based on the byte-serial AES implementation [12]. It has 56-bit and 64-bit shift registers for temporarily storing incoming bytes and feeding them to the AES implementation synchronously. The design has a byte-oriented interface: an 8-bit input port for feeding the associated data A, nonce N, and message M, (ii) another 8-bit input port for the AES key K, and (iii) an 8-bit output port for the ciphertext C and tag T.

Table 5 and Table 6 summarize the post-synthesis performances on ASIC and FPGA, respectively. The tables show the circuit area, maximum frequency, and latency of AES, SAEAES, and their difference.

The ASIC performance in Table 5 is evaluated with the NanGate 45-nm CMOS standard cell library [1]. SAEAES is implemented with 3,502 [GE]: 823 [GE] in addition to 2,679 [GE] for the AES implementation. The SAEAES implementation consumes a message block in 231 cycles which is the same as the latency of the underlying AES implementation. It means that the additional operations for SAEB can be finished in parallel to the AES implementation.

The FPGA performances in Table 6 are evaluated for Xilinx Virtex-7 and Intel Cyclone V FPGAs. Similarly to the ASIC implementation, the SAEAES implementation can be realized with minimal overhead in addition to the AES implementation.

5.3 Resistance against Side-Channel and Fault Injection Attacks

SAEAES is composed of AES and some XOR operations. Therefore, it is reasonable to assume that the resistance against side-channel and fault injection attacks is determined by the underlying AES implementation. AES is the most popular block cipher algorithm for studying countermeasures against side-channel and fault injection attacks. Threrefore, implementers can choose appropriate countermeasures against side-channel and fault injection attacks for AES to protect SAEAES.

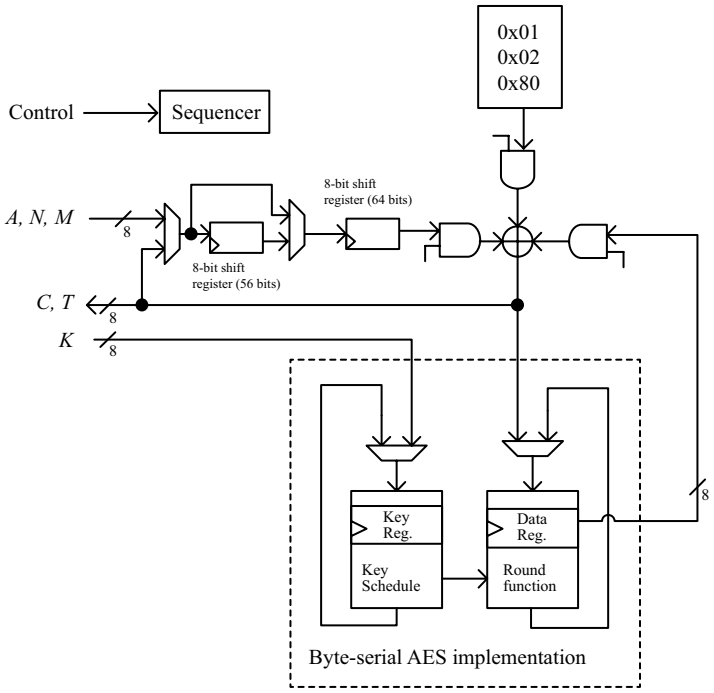

Fig. 4. A lightweight hardware architecture of `SAEAES` (Figure 4 of [13]).

Table 5. Performance of `SAEAES` in *NanGate* 45-nm CMOS standard cell library.

Target	Circuit Area [GE]	Max. Freq. [MHz]	Latency [Cycles]
`SAEAES`	3,502	122.0	231
`AES`	2,679	126.4	231
diff.	823	—	—

Table 6. Performance of `SAEAES` in FPGA.

Platform	Target	Look-up Table [LUTs/ALMs]	Flip-flop [FFs]	Max. Freq. [MHz]	Latency [Cycles]
Xilinx Virtex-7	`SAEAES`	348	242	145.9	231
`xc7vx330t`	`AES`	304	218	144.2	231
`ffg1157-1`	diff.	44	24	—	—
Intel Cyclone V	`SAEAES`	299	410	83.3	231
`5CEBA2F17C8`	`AES`	264	283	87.8	231
	diff.	35	127	—	—

6 Third Party Analyses

The original paper of SAEB is reviewed and published in the IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES) [13].

References

[1] NanGate Open Cell Library. http://www.nangate.com/.

[2] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the Indifferentiability of the Sponge Construction. In *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 181–197. Springer, 2008.

[3] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications. In *SAC 2011*, volume 7118 of *LNCS*, pages 320–337. Springer, 2012.

[4] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Permutation-based encryption, authentication and authenticated encryption. In *DIAC 2012*, 2012.

[5] Alex Biryukov and Dmitry Khovratovich. Related-Key Cryptanalysis of the Full AES-192 and AES-256. In *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 1–18. Springer, 2009.

[6] Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolic. Distinguisher and Related-Key Attack on the Full AES-256. In *CRYPTO 2009*, volume 5677 of *LNCS*, pages 231–249. Springer, 2009.

[7] Andrey Bogdanov, Donghoon Chang, Mohona Ghosh, and Somitra Kumar Sanadhya. Bicliques with minimal data and time complexity for AES. In *Information Security and Cryptology - ICISC 2014 - 17th International Conference, Seoul, Korea, December 3-5, 2014, Revised Selected Papers*, volume 8949 of *LNCS*, pages 160–174. Springer, 2014.

[8] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique Cryptanalysis of the Full AES. In *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 344–371. Springer, 2011.

[9] Patrick Derbez, Pierre-Alain Fouque, and Jérémy Jean. Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting. In *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 371–387. Springer, 2013.

[10] Leibo Li, Keting Jia, and Xiaoyun Wang. Improved Single-Key Attacks on 9-Round AES-192/256. In *FSE 2014*, volume 8540 of *LNCS*, pages 127–146. Springer, 2014.

[11] Rongjia Li and Chenhui Jin. Meet-in-the-middle attacks on 10-round AES-256. *Des. Codes Cryptography*, 80(3):459–471, 2016.

[12] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 69–88. Springer, 2011.

[13] Yusuke Naito, Mitsuru Matsui, Takeshi Sugawara, and Daisuke Suzuki. SAEB: A Lightweight Blockcipher-Based AEAD Mode of Operation. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):192–217, 2018.

[14] Chanathip Namprempre, Phillip Rogaway, and Thomas Shrimpton. Reconsidering Generic Composition. In *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 257–274. Springer, 2014.

[15] Morris Dworkin (NIST). Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, NIST Special Publication 800-38D, November 2007.

[16] Morris Dworkin (NIST). Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality, NIST Special Publication 800-38C, May 2004.

[17] National Institute of Standards and Technology (NIST). Advanced Encryption Standard (AES). FIPS PUB 197. November 26, 2001).

[18] Phillip Rogaway. Authenticated-encryption with associated-data. In *CCS 2002*, pages 98–107. ACM, 2002.

[19] Phillip Rogaway and Thomas Shrimpton. A Provable-Security Treatment of the Key-Wrap Problem. In *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 373–390. Springer, 2006.

[20] Biaoshuai Tao and Hongjun Wu. Improving the Biclique Cryptanalysis of AES. In *ACISP 2015*, volume 9144 of *LNCS*, pages 39–56. Springer, 2015.