# SPAE a mode of operation for AES on low-cost hardware *

Philippe Elbaz-Vincent[1], Cyril Hugounenq[1] and Sébastien Riou[2]

[1] Univ. Grenoble Alpes, CNRS, IF, 38000 Grenoble, France,
philippe.elbaz-vincent@univ-grenoble-alpes.fr,
cyril.hugounenq@univ-grenoble-alpes.fr

[2] Tiempo, 38330 Montbonnot Saint Martin, France, sebastien.riou@tiempo-secure.com

**Abstract.** We propose SPAE, a single pass, patent free, authenticated encryption with associated data (AEAD) for AES. The algorithm has been developped to address the needs of a growing trend in IoT systems: storing code and data on a low cost flash memory external to the main SOC. Existing AEAD algorithms such as OCB, GCM, CCM, EAX , SIV, provide the required functionality however in practice each of them suffer from various drawbacks for this particular use case. Academic contributions such as ASCON and AEGIS-128 are suitable and efficient however they require the development of new hardware accelerators and they use primitives which are not 'approved' by governemental institutions such as NIST, BSI, ANSSI. From a silicon manufacturer point of view, an efficient AEAD which use existing AES hardware is much more enticing: the AES is required already by most industry standards invovling symmetric encryption (GSMA, EMVco, FIDO, Bluetooth, ZigBee to name few). This paper expose the properties of an ideal AEAD for external memory encryption, present the SPAE algorithm and analyze various security aspects. Performances of SPAE on actual hardware are better than OCB, GCM and CCM.

**Keywords:** authenticated encryption with associated data (AEAD), · Nonce Misuse Resilience · Execute in Place (XIP) · Differential Fault Analysis (DFA) · AES · low-cost hardware

## 1 Introduction and motivations for AEAD on embedded systems

In the past, most embedded systems would store everything within the internal flash memory of a microcontroller. As performance and memory requirements increase, the trend is to use powerful application processors coupled with discrete flash memory chips. This has great impact on the security of the application and, counter-intuitively, even when physical attacks are not considered a threat. In a connected world, remote attacks are typically the most devastating ones so most manufacturers focus on them. The catch is that remote attacks are possible only once a weakness has been found. As a result attackers sometimes perform a physical attack as a preliminary step for finding an exploit [20].

The firmware was stored in a secure internal flash and its confidentiality was a one time problem: once the firmware was decrypted, it was stored in plain in the internal memory. Similarly authenticity had to be verified just once before activating a new version of the

*This work is supported by SECURIOT-2-AAP FUI 23, ANR-15-IDEX-02 and partially supported by ANR-15-CE39-0002.

firmware. In that setting, the performance of cryptographic operations have no impact on the performance of the application and very little impact on the battery.

Now that the firmware is stored in external flash, guaranteeing its authenticity and confidentiality is much more challenging. Those functionalities are typically provided by the use of an authenticated-encryption with associated-data (AEAD) algorithm and a secret key stored on-chip. Furthermore the code is executed in XIP mode (eXecute In Place). This means that the application processor fetches code from the external memory in much the same way as it would do with an internal memory: it fetches small blocks to fill just one cache line at a time (typically 256 bytes). The performance of the AEAD therefore impact directly the application performance and the battery life time. In this case, the AEAD or at least the underlying block cipher is implemented in hardware.

Besides XIP, a number of microcontroller application also store data in external flash simply because they need more capacity than the internal flash has. In this case the performance of the AEAD is less critical but the need for energy efficiency remains. The AEAD may be implemented fully in software or with some hardware acceleration. In this context it is desirable to use AES as the core function:

- AES is currently the only symmetric encryption primitive 'approved' by NIST, BSI and ANSSI (Triple DES will be retired in 2023) [19].
- Numerous microcontrollers have AES hardware accelerator built-in. For example STM32L083RB, LPC18S37JET100, PIC24FJ128GA202, nRF52840.
- In secure elements, the AES accelerator is heavily protected against physical attacks. There is simply no way of achieving a better trade off between perfomance, security and power consumption on this type of chips.

*Remark* 1. The statement about the efficiency of AES on secure elements is just an observation of what is typically available on those ICs at the time of writing. For the foreseenable future, low cost means reusing AES as far as smart cards and secure elements are concerned.

Ideally the AEAD would lend itself to efficient protection against side channel attacks. This means to avoid using the addition operation, so ARX schemes shall be avoided [10]. In other words, besides AES, the AEAD shall use only operations which are trivial to mask such as XOR and rotations.

As a result, the properties of the ideal AEAD for external memory encryption are the following:

- Energy efficiency on small message size (full blocks, 256 bytes typically)
- Fast in single thread setting
- Differential Fault Analysis (DFA) resistance at algorithmic level
- Use only AES, XOR and rotate
- Efficient both in software and hardware implementations against comparable AEAD

*Remark* 2. A common misconception about AEAD algorithms is the belief that they are intrisically protected against DFA ([11]). This is not the case for GCM for example: the tag is computed from the ciphertext so faults injected on AES during decryption are not detected. One may object that DFA requires the observation of the faulty output and that the plaintext is available only inside the device. This is certainly the general case however one cannot assume that plaintext is never sent outside of the device. For example it is common for firmware to send data such as version numbers unencrypted. This is especially true of applications with graphic display. We make this point on DFA because the typical countermeasure doubles the execution time and energy consumed by a block cipher. The intrinsic DFA resistance is therefore a desirable property.

The main contribution of this paper is to show that SPAE, which is patent-free, fulfills **all** those properties.

First we present a state of the art (section 2), after a description of SPAE (section 3), a security analysis of the scheme is done (section 4). Then we evaluate its performances (section 5), we conclude with perspectives and future improvements of our work (section 6). Finally in the appendix (section 7) there are some test vectors and some design rationale.

In particular, for the security analysis (section 4), we provide results about differential fault analysis, nonce misuse resilience([3, 4]) authenticity and privacy (theorem 1 and 2) and arguments on the resilience of our scheme against an attack like the one of Inoue *et al.* [24, 35, 25, 23]. (section 4.1.5).

# 2 State of the art on AEAD mode of operation

An authenticated encryption (AE) scheme is a symmetric-key mechanism with the goal that the ciphertext protect the privacy and the authenticity of the plaintext. In real-world applications, not all data should be privacy-protected and it gave rise to the notion of authenticated-encryption with associated-data (AEAD), which was first introduced by Rogaway [36]. Since then several families of AEAD schemes have been proposed and standardized by NIST, IEEE, IETF and ANSI (C12.22). An overview of the state on AE and AEAD schemes, with discussions of security and privacy aspects, has been given by Vizár [44]. In practice, we distinguish two kinds of AEAD schemes (idem for AE schemes) [6]. In a *two-pass* scheme we make two passes through the data, one aimed at providing privacy and the other, authenticity. In a *one-pass* AEAD or AE scheme we make a single pass through the data, simultaneously doing what is needed to engender both privacy and authenticity. As we can expect, the computational cost is usually about half that of a two-pass scheme.
As part of the standard hypothesis, it is considered to be the responsibility of the sender not to reuse any nonce (often by using a counter). Nevertheless, nonce can still get reused in AE and AEAD schemes, either due to implementation errors or sophisticated attacks. Rogaway and Shrimpton [40], introduced the notion of nonce-misuse resistant AE and AEAD (MRAE) with an associated security model. Their work was seminal for the design of new AE and AEAD schemes that are robust to improper use or implementation errors. In the following we list the most frequent AEAD algorithms used in industry and comment on their energy efficiency and nonce-misuse, assuming the underlying block cipher is AES.

## 2.1 NIST approved AEAD algorithms

NIST [1] has approved two AEAD algorithms: AES-CCM (800-38C) and AES-GCM (800-38D).

AES-CCM [18] is a two pass 'authenticate-then-encrypt' scheme. As such it is bound to be about twice slower as a single pass scheme [42]. CCM has been criticized as not being 'on-line' [6]: the length of the message must be known at the beginning of the processing. This is not a problem in the context of external memory encryption.

AES-GCM [30] is also a two pass scheme however the authentication pass use a dedicated operator 'GHASH'. As it it shown in section 5 the GHASH operation is typically slower than AES-128 when implemented in software on microcontrollers. As a result, in that use case at least, GCM is less power efficient than CCM.

[1] https://csrc.nist.gov/projects/block-cipher-techniques/bcm/current-modes

## 2.2 CAESAR finalists

The CAESAR competition [2] produced many alternatives to the standard AEAD algorithms, unfortunately only two among them reuse the full AES: COLM and OCB.

COLM by Andreeva *et al.* [2] consists of two layers of encryption connected by a linear mixing layer. Its energy efficiency is therefore similar or worse than classic two pass AEAD such as CCM.

OCB by Rogaway *et al.* [28] is a single pass scheme (it is the third version there was [38, 37]). As such it is intrinsically more energy efficient than two pass schemes. It has been extensively reviewed by the cryptographic community and has been adopted in few internet standards. It appears as the most suitable algorithm for external memory encryption however a non technical issue hampers its use in the industry: it is patented.

## 2.3 Other prominent AEAD algorithms

Many other AEAD algorithms have been proposed, two stands out: SIV and CHACHA20-POLY1305. The SIV mode by Rogaway and Shrimpton [41] attracted a lot of attention due to its advantage of keeping strong confidentiality guarantees in the event of nonce reuse. It is nevertheless a two pass scheme and therefore is not suitable with respect to our energy efficiency constraint.

The schemes CHACHA20 and POLY1305 both by Bernstein [8] [7] are often combined to provide authenticated encryption. They are praised for their high efficiency in software implementations. As CHACHA20 is an ARX scheme, it is difficult to protect against side channel attacks [10]. All the efficiency advantage that CHACHA20 has over AES are likely to vanish after the addition of side channel countermeasures so we do not consider it further.

The mode of operation SAEB by Naito *et al.* [31] is a single pass scheme but has recommended parameters that makes it equivalent to a two pass scheme and since it is patented it is not used in the industry.

## 2.4 Comparison between the AEAD schemes

We summarize below the properties and operations count for the different schemes aforementioned which use AES. We include SPAE and CSPAE, our proposal, for comparison.

**Table 1:** Comparison of selected AEAD schemes

| Name | Non trivial operations count | Consequence of Nonce reuse |
|---|---|---|
| EAX [6] | (2m+a+4)Ek | Xor of plaintexts revealed |
| CCM [18] | (2m+a+2)Ek | Xor of plaintexts revealed |
| SAEB [31] | (2m+a+2)Ek (with parameters used in [31, § 6]) | Equality of first blocks revealed |
| SIV [41] | (2m+a+1)Ek | Equality of message revealed |
| CLOC [26] | (2m+a+1)Ek | Equality of blocks revealed |
| GCM [30] | (m+1)Ek + (m+a+1)GHASH | • Forgeability<br>• Xor of plaintexts revealed |
| OCB [28] | (m+a+2)Ek + (m+a+1)Inc | • Forgeability<br>• Equality of blocks revealed |
| SPAE | (m+a+2)Ek | Equality of first blocks revealed |
| CSPAE | (m+a+2)Ek | Equality of first blocks revealed |

[2] https://competitions.cr.yp.to/caesar.html, final portfolio announced February, 20, 2019.

Notations: m (length of message, in blocks), a (length of associated data, in blocks), Ek (the encryption scheme with the key k), GHASH (multiplication in $\mathbb{F}_{128}$), Inc (the 'inc' operation defined in OCB).

In the next section we will present the SPAE and CSPAE schemes and will show that they are the fastest modes in single thread setting. In case of nonce reuse, only SIV provides better confidentiality.

# 3 The SPAE and CSPAE modes of operations

In order to combine energy efficiency and protection against improper use or implementation errors, our new scheme will need to be one-pass and provide some level of nonce-misuse resistance (no forgeability, no key recovery, see section 4.1.6). We will describe it for a general block cipher even if we have for application AES in mind. Furthermore, only elementary binary operations should be used in order to contain the energy consumption and facilitate the countermeasures. We will start by an outline of the basic operations needed by the scheme.

## 3.1 Elementary Operations

We denote by $\overline{x}$ the bitwise complement of $x$, $\oplus$ is the bitwise XOR and $\wedge$ is the bitwise AND. We denote by $a << b$ the left shifting of $a$ by $b$ bits, the least significant bits being filled with 0, we denote $a||b$ as the concatenation of $a$ and $b$. We write integer variable names using this style: *intname*. We write block names using this style: `BLOCKNAME`. $HSWAP(a)$ denotes the swap of the left half with the right half within a block.

## 3.2 Inputs and outputs of the scheme

We denote by $bs$ the bit size of blocks used in this scheme, it is $\geqslant 128$. $ml$ and $adl$ represent the message (resp. associated data) length in bits. $P$ is the plaintext, it is composed of $m = \frac{(ml+bs-1)}{bs}$ blocks $\mathtt{P}_i$ of size $bs$. $A$ is the associated data used to authenticate the ciphered text, it is composed of $a = \frac{(adl+bs-1)}{bs}$ blocks $\mathtt{A}_i$ of size $bs$. $C$ is the ciphertext, it is composed of $m$ blocks $\mathtt{C}_i$ of size $bs$. `TAG` is the authentication tag generated for the message, it is of bit size $ts \leqslant bs$. $\mathtt{TAG}_{null}$ is the authentication tag generated for an empty message with associated data. `NONCE` is a value used only once for initiating SPAE encryption of size $bs$. `K` is the secret key used of size $bs$. $E_{\mathtt{K}}$ is the cryptographic primitive used in the scheme it should be able to take as input a key `K`, a block of size $bs$ and outputs a block of size $bs$.

## 3.3 Internal data of the scheme

We use a total of 9 internal variables of size $bs$. Three variables $\mathtt{PT}_i, \mathtt{CT}_i$ and `MT` for the encryption part of the scheme. The first two are intialised before the encryption of the message and updated during this one and `MT` is produced at the end of the encryption of the message. We use the variables $\mathtt{AT}_i$ in order to update the processing of associated data. `IT` and `PADINFO` are used for the production of the `TAG`, `KN` is a secret key derived from `K` used all along the encryption part and the production of the `TAG` and $\mathtt{1_{bs}}$ is a constant with all bits set to 1. As a result $x \oplus \mathtt{1_{bs}} = \overline{x}$

## 3.4 Diagrammatic description of SPAE and CSPAE

The two schemes only differ by the computation of `KN`, $\mathtt{PT}_0$ and $\mathtt{CT}_0$. The below diagrams don't show those computations and apply equally to both schemes. The figure 1 illustrates the encryption of a message with three blocks of plaintext and three blocks of associated data. The decryption of such cyphertext is shown in figure 2. In figure 3 is shown the authentication of two blocks of associated data.

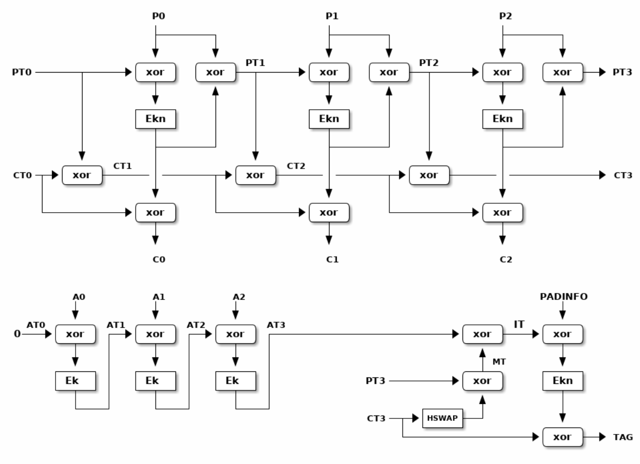


**Figure 1:** Encryption with SPAE with $a = 3$ and $m = 3$

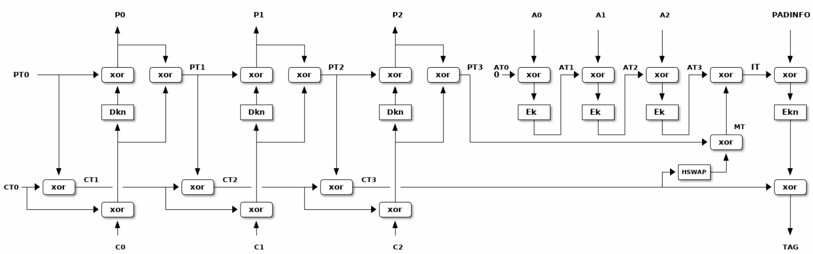


**Figure 2:** Decryption with SPAE with $a = 3$ and $m = 3$

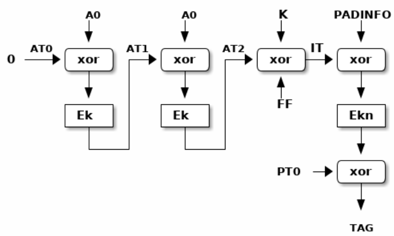


**Figure 3:** Authentication of two blocks of associated data with SPAE

## 3.5 Algorithmic description of SPAE and CSPAE

Both algorithms require an initialization state which is different. CSPAE avoids to call $E$ with different keys and could be considered as a tweakable block cipher as defined in

[29] whereas SPAE does not fit this definition. This is done at the expense of turning the static call to $E$ into a dynamic one, i.e. a call which depends on $\mathtt{NONCE}$. Further details on the choices made for the initial values are in section 7.4. Only the 'Init' algorithm differs between SPAE (algorithm 1) and CSPAE (algorithm 2).

**Algorithm 1** $\mathtt{Init}_{\mathtt{SPAE}}$ (SPAE version)

**Require:** $\mathtt{K}$ the secret key, $\mathtt{NONCE}$ a nonce
**Ensure:** $\mathtt{PT}_0$ and $\mathtt{CT}_0$and $\mathtt{KN}$

$\mathtt{CT}_0 \leftarrow E_{\mathtt{K}}(\mathtt{K})$
$\mathtt{PT}_0 \leftarrow \mathtt{K} \oplus \mathtt{CT}_0$
$\mathtt{KN} \leftarrow \mathtt{K} \oplus \mathtt{NONCE}$
**return** $(\mathtt{PT}_0, \mathtt{CT}_0, \mathtt{KN})$

**Algorithm 2** $\mathtt{Init}_{\mathtt{CSPAE}}$ (CSPAE version)

**Require:** $\mathtt{K}$ the secret key, $\mathtt{NONCE}$ a nonce
**Ensure:** $\mathtt{PT}_0$ and $\mathtt{CT}_0$and $\mathtt{KN}$

$\mathtt{CT}_0 \leftarrow E_{\mathtt{K}}(\mathtt{NONCE} \oplus \mathtt{K})$
$\mathtt{PT}_0 \leftarrow \mathtt{NONCE} \oplus \mathtt{K} \oplus \mathtt{CT}_0$
$\mathtt{KN} \leftarrow \mathtt{K}$
**return** $(\mathtt{PT}_0, \mathtt{CT}_0, \mathtt{KN})$

Now we present algorithms used to process input data and produce output.

**Algorithm 3** ComputeTag

**Require:** $a \geqslant 0, \mathtt{A}_0 ... \mathtt{A}_{a-1}$ associated data blocks of size $bs$, $\mathtt{K}$ the secret key, $\mathtt{PT}_m$, $\mathtt{CT}_m$ and $\mathtt{KN}$
**Ensure:** $\mathtt{TAG}$

$\mathtt{AT} \leftarrow 0$
**for** $i \in 0, \cdots, a-1$ **do**
  $\mathtt{AT} \leftarrow E_{\mathtt{K}}(\mathtt{AT} \oplus \mathtt{A}_i)$
**end for**
**if** $m = 0$ **then**
  $\mathtt{MT} \leftarrow \mathtt{1}_{\mathtt{bs}} \oplus \mathtt{K}$
**else**
  $\mathtt{MT} \leftarrow HSWAP(\mathtt{CT}_m) \oplus \mathtt{PT}_m$
**end if**
$\mathtt{PADINFO} \leftarrow \mathtt{PadInfo}(ml, adl)$
$\mathtt{IT} \leftarrow \mathtt{AT} \oplus \mathtt{MT}$
$\mathtt{TAG} \leftarrow E_{\mathtt{KN}}(\mathtt{IT} \oplus \mathtt{PADINFO})$
**if** $m = 0$ **then**
  $\mathtt{TAG} \leftarrow \mathtt{TAG} \oplus \mathtt{PT}_m$
**else**
  $\mathtt{TAG} \leftarrow \mathtt{TAG} \oplus \mathtt{CT}_m$
**end if**
**return** $\mathtt{TAG}$

The function $\texttt{PadInfo}(ml, adl)$ is computed as below(in the applied case of $bs = 128$):

$$\begin{aligned}
(ml) &\mapsto mp = ml \wedge ((1 << 64) - 1) \\
(adl) &\mapsto adp = adl \wedge ((1 << 64) - 1) \\
(mp) &\mapsto mp_{32} = mp \wedge ((1 << 32) - 1) \\
(adp) &\mapsto adp_{32} = adp \wedge ((1 << 32) - 1) \\
(mp, adp, adp_{32}) &\mapsto \texttt{TMP} = ((adp >> 32) \oplus (adp_{32} << 32)) \oplus mp \\
(mp_{32}, adp_{32}, \texttt{TMP}) &\mapsto \texttt{PADINFO} = (\texttt{TMP} << 64) \oplus (adp_{32} << 32) \oplus mp_{32}
\end{aligned}$$

We now give a description of the two main algorithms 4 and 5 for encryption and decryption. The subscript $s$ denotes the choice between the schemes SPAE and CSPAE.

**Algorithm 4** $\texttt{EncryptAndAuthenticate}_s$

**Require:** $P$ a stream of message blocks, $\texttt{K}$ a secret key, $\texttt{NONCE}$ a nonce,and $A$ a stream of associated data blocks
**Ensure:** $(C, \texttt{TAG})$ a stream of ciphered blocks followed by $\texttt{TAG}$;

```
(PT, CT, KN) ← Init_s(NONCE, K)
for i ∈ 0, ··· , m − 1 do
  TMP ← E_KN(PT ⊕ P_i)
  C_i ← TMP ⊕ CT
  CT ← CT ⊕ PT
  PT ← P_i ⊕ TMP
end for
TAG ← ComputeTag(A, K, PT, CT, KN)
return (C_0||...||C_{m−1}, TAG)
```

**Algorithm 5** $\texttt{DecryptAndAuthenticate}_s$

**Require:** $C$ a stream of ciphered blocks, $\texttt{TAG}$, and the values used to produce them: the key $\texttt{K}$, the nonce $\texttt{NONCE}$ and the associated data $A$;
**Ensure:** The message $P$ or *Failure*

```
(PT, CT, KN) ← Init_s(NONCE, K)
for i ∈ 0, ··· , m − 1 do
  TMP ← CT ⊕ C_i
  CT ← CT ⊕ PT
  P_i ← PT ⊕ D_KN(TMP)
  PT ← P_i ⊕ TMP
end for
TAGTEST ← ComputeTag(A, K, PT, CT, KN)
if TAGTEST = TAG then
  return (P_0||...||P_{m−1})
end if
return Failure
```

Like in any AEAD algorithm, the decryption shall check equality of $\texttt{TAGTEST}$ and $\texttt{TAG}$ before returning any $\texttt{P}_i$. If the test fails, it shall return nothing besides *Failure*.

## 3.6 Padding

Padding is supported for both the message and the associated data. During encryption, the last block is padded by appending '0' bits until $bs$ is reached. The original length informations are encoded in PADINFO which contributes to the computation of TAG. This TAG would be the only difference between the processing of one message padded by the algorithm and one by the user. Note that although the length informations are processed by the algorithm, they are not recoverable from the ciphertext. This means that the decryption algorithm expects to be called with correct length informations and that it is up to the user to ensure a method of doing so.

### 3.7 Design Rationale

The purpose of the $HSWAP$ operation and the redundant encoding of PADINFO is to prevent DFA attack on the $D_{kn}$ operations during decryption (see 4.2).

The $\mathtt{TAG}_{null}$ equation is required in order to avoid the 0 value for the TAG. Indeed the $HSWAP$ operation is transparent to all symmetric values, in those cases and if $a = 0$ and $m = 0$, the TAG equation for the general case would then lead to 0. It is therefore needed to have a special equation for $\mathtt{TAG}_{null}$:

$$\begin{aligned}
\mathtt{TAG}_{null} &= \mathtt{PT}_0 \oplus E_{\mathtt{KN}}(\mathtt{AT}_0 \oplus \mathtt{K} \oplus \mathtt{FF} \oplus \mathtt{PADINFO}) \\
&= \mathtt{K} \oplus E_{\mathtt{K}}(\mathtt{K}) \oplus E_{\mathtt{K}\oplus\mathtt{NONCE}}(\mathtt{K} \oplus \mathtt{FF}) && (SPAE) \\
&= \mathtt{NONCE} \oplus \mathtt{K} \oplus E_{\mathtt{K}}(\mathtt{K} \oplus \mathtt{NONCE}) \oplus E_{\mathtt{K}\oplus\mathtt{NONCE}}(\mathtt{K} \oplus \mathtt{FF}) && (CSPAE)
\end{aligned}$$

Moreover the value of the $\mathtt{TAG}_{null}$ being computed with a secret value used only here ($\overline{k} = \mathtt{K} \oplus \mathtt{1}_{\mathtt{bs}}$), permits to not give to the attacker new relations between the $\mathtt{TAG}_{null}$ and other variables of the scheme in other mode.

The recommended value for $ts$ is $bs$. Implementations are free to reduce $ts$ by selecting any part of the TAG, all selections of a given number of bits will result in the same security level.

As in all the usages for AEAD, there is no method of early abort implemented in SPAE. This choice protects SPAE against side channel attacks like the one of [1].

## 4 Security analysis of SPAE

First we recall some classical definitions. We denote by $\mathcal{K}$ a finite space called the key space, $\mathcal{N}$ a finite space called the nonce space, $\mathcal{M} \subset \{0,1\}^*$ the message space, $\mathcal{C} \subset \{0,1\}^*$ the ciphertext space and, $\mathcal{A} \subset \{0,1\}^*$ the associated data space. A *block cipher* is a deterministic algorithm $E : \mathtt{K} \times \{0,1\}^{bs} \to \{0,1\}^{bs}$ with $bs \geqslant 1$. We denote $E_{\mathtt{K}}(.) = E(\mathtt{K}, .)$ and we require that it is a permutation for all $\mathtt{K} \in \mathcal{K}$. Let $D = E^{-1}$ be the map from $\mathcal{K} \times \{0,1\}^{bs}$ to $\{0,1\}^{bs}$ defined by $D_{\mathtt{K}}(Y) = D(\mathtt{K}, Y) = X$ with $X$ being the unique point such that $E_{\mathtt{K}}(X) = Y$.

The encryption algorithm $\mathcal{E}$ takes a tuple $(\mathtt{K}, N, A, M) \in \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M}$ and returns deterministically, either a ciphertext $C = \mathcal{E}_{\mathtt{K}}^{N,A}(M) \in \mathcal{C} \subset \{0,1\}^*$ or the distinguished value *Failure*. We denote by $a$ the number of blocks $\mathtt{A}_i$ of associated data and by $m$ the number of blocks $\mathtt{P}_i$ of message.

The decryption algorithm $\mathcal{D}$ takes a tuple $(\mathtt{K}, N, A, C) \in \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{C}$ and returns deterministically, either *Failure* or a string $M = \mathcal{D}_{\mathtt{K}}^{N,A}(C) \in \mathcal{M} \subset \{0,1\}^*$. We require that $\mathcal{D}_{\mathtt{K}}^{N,A}(C) = M$ for any string $C = \mathcal{E}_{\mathtt{K}}^{N,A}(M)$ and that $\mathcal{E}$ and $\mathcal{D}$ return *Failure* if they are provided an input outside of $\mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M}$ or $\mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{C}$. We require that $|\mathcal{E}_{\mathtt{K}}^{N,A}(M)| = |\mathcal{E}_{\mathtt{K}}^{N,A}(M')|$ when $|M| = |M'|$, when the value of $|\mathcal{E}_{\mathtt{K}}^{N,A}(M)| = |M| + \tau$ we call $\tau$ the tag length of the scheme.

We denote $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ an AE scheme. Given an adversary $\mathcal{A}$, we denote, following [29, 2.2], [3, Definition 1,2] and [4], the resilience advantage $\mathtt{Adv}_{\Pi}^{priv}(\mathcal{A}) = \Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathcal{E}_K(.,.,.)} \Rightarrow 1] - \Pr[\mathcal{A}^{\$(.,.,.)} \Rightarrow 1]$ where queries of $\$(N, A, M)$ return a uniformly random string of length $|\mathcal{E}_K^{N,A}(M)|$. $\mathcal{A}$ is allowed to asks queries with the same $\mathtt{NONCE}$ (i.e. the same first component) but the attacker should respect the $\mathtt{NONCE}$ uniqueness only when the attacker tries to distinguish $\mathcal{E}_K$ from \$. The attacker is also not allowed to ask a query outside of $\mathcal{N} \times \mathcal{A} \times \mathcal{M}$. Repeating a query isn't allowed too. Authenticity resilience, is denoted by $\mathtt{Adv}_{\Pi}^{auth} = \Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathcal{E}_K(.,.,.)} \textit{forges}]$ where *forges* means that the adversary outputs a value $(N, A, C) \in \mathcal{N} \times \mathcal{A} \times \mathcal{C}$ such that $\mathcal{D}_K^{N,A}(C) \neq \textit{Failure}$ and there was no prior request $(N, A, M)$ such that it returned $C$ and $N$ has not been used twice with $\mathcal{E}_K(.,.,.)$.

The nonce resilience is a notion to ensure that encryption/decryption done with different nonces are independant sucht that the information obtained with nonce reused for a certain $\mathtt{NONCE}$ is not useful for another $\mathtt{NONCE}$.

*Remark* 3. We will make the assumption all along this document that the $\mathtt{NONCE}$ used are known and chosed by the attacker, the reuse of $\mathtt{NONCE}$ are limited as stated above.

We denote by $\sigma_e$ the total number of request to $E_{\mathtt{KN}}$ in the scheme to process message blocks, $\sigma_a$ the total number of request to $E_{\mathtt{K}}$ in the scheme to process associated data blokcs. We have for SPAE/CSPAE $\sigma_a + \sigma_e + 2$ requests for the encryption functions $E_{\mathtt{KN}}, E_{\mathtt{K}}$ for the requests $(N^i, A^i, M^i)$. The two extra calculations are the computation for the $\mathtt{TAG}$ and $\mathtt{CT}_0$, this last one is always the same for SPAE and could be computed only once.

First we do a structural analysis of SPAE (4.1) to show that the schemes reveals no information about the blockcipher and we evaluate its authenticity(theorem 1). Then we give a Differential Fault Analysis (4.2) and we conclude with an analysis of the privacy (4.4).

## 4.1 Structural analysis of SPAE

A cryptanalysis of the scheme would start by trying to get the key $\mathtt{K}$ which is direct from $\mathtt{KN}$. Therefore having the more information available about the use of $E_{\mathtt{KN}}$ is a prerequisite to cryptanalyse this scheme who could present some flaw since it uses only XOR operations outside of the usage of a blockcipher $E$.

We will therefore show what information we could get from the structure of the scheme and the relation between the variables used.

### 4.1.1 Analysis of the internal variables (Passive attacks)

Since the internal variables $\mathtt{PT}_i, \mathtt{CT}_i$ are used to mask the inputs and outputs of $E_{\mathtt{KN}}$, it is relevant to see how they are related, how we can get their values and what information we could get from them.

This analysis is made in a context of some passive attack where the attacker just put in input plaintexts $(\mathtt{P}_i)$ and associated datas $(\mathtt{A}_i)$ and observe the outputs $\mathtt{C}_i, \mathtt{TAG}$.

**Proposition 1.** *Under the assumption that the attacker has access to all the* $\mathtt{P}_i, \mathtt{C}_i$*;*

- *the knowledge of* $\mathtt{PT}_j$ *and* $\mathtt{CT}_{j-1}$ *are equivalent,*
- *the knowledge of* $\mathtt{PT}_j$ *and* $\mathtt{CT}_{j-1}$ *does not bring more knowledge about the internal variables of the scheme.*

*Proof.* The first claim is a consequence of the following equalities $\mathtt{CT}_{j-1} = \mathtt{C}_{j-1} \oplus E_{\mathtt{KN}}(\mathtt{P}_{j-1} \oplus \mathtt{PT}_{j-1}) = \mathtt{C}_{j-1} \oplus \mathtt{PT}_j \oplus \mathtt{P}_{j-1}$.

The proof is straightforward and consists to explore the relations we have between the internal variables and show that we got to know at some points internal variables such as $\mathtt{CT}_{j-3}, \mathtt{CT}_{j+1}, \mathtt{PT}_{j+2}, \mathtt{PT}_{j-2}$ which we could not have a hint on their values from the assumptions made. □

*Remark* 4. If we know $\mathtt{PT}_j, \mathtt{CT}_{j+1}$ then we can compute $\mathtt{CT}_j = \mathtt{PT}_j \oplus \mathtt{CT}_{j+1}$.

Therefore we study here the knowledge of a pair of values $\mathtt{PT}_i, \mathtt{CT}_i$.

**Proposition 2.** *Under the assumption that the attacker knows all the* $\mathtt{P}_i, \mathtt{C}_i$ *for* $i \in [0, m]$*, if the attacker knows* $\mathtt{PT}_r, \mathtt{CT}_r$ *for some* $r \in [0, m]$ *then the attacker can deduce all possible value* $\mathtt{PT}_j, \mathtt{CT}_j$*. Therefore the attacker is able to compute* $\mathtt{CT}_1 = \mathtt{KN} \oplus \mathtt{NONCE}$*.*

*Proof.* From $\mathtt{C}_r = \mathtt{CT}_r \oplus E_{\mathtt{KN}}(\mathtt{P}_r \oplus \mathtt{PT}_r)$ we get the value $E_{\mathtt{KN}}(\mathtt{P}_r \oplus \mathtt{PT}_r)$ from it we deduce $\mathtt{PT}_{r+1} = E_{\mathtt{KN}}(\mathtt{P}_r \oplus \mathtt{PT}_r) \oplus \mathtt{P}_r$, we get $\mathtt{CT}_{r+1} = \mathtt{PT}_r \oplus \mathtt{CT}_r$. At this point we get the pair of values $\mathtt{PT}_{r+1}, \mathtt{CT}_{r+1}$. Now we look for the preceding values, we got $E_{\mathtt{KN}}(\mathtt{P}_{r-1} \oplus \mathtt{PT}_{r-1}) = \mathtt{PT}_r \oplus \mathtt{P}_{r-1}$, we then deduce $\mathtt{CT}_{r-1} = \mathtt{C}_{r-1} \oplus E_{\mathtt{KN}}(\mathtt{P}_{r-1} \oplus \mathtt{PT}_{r-1})$, we now got the preceding couple $\mathtt{PT}_{r-1} = \mathtt{CT}_{r-1} \oplus \mathtt{CT}_r, \mathtt{CT}_{r-1}$. We apply those knowledge recursively to get all the pairs $\mathtt{PT}_\ell, \mathtt{CT}_\ell$. □

### 4.1.2 Analysis of collection of tuples $(X, E_{\mathtt{K}}(X))$

Since any attack against the key K would start with collecting images of the application of $E_{\mathtt{KN}}$ on some constants known and sometimes chosen, we first start with some remarks studying the security of the cryptosystem against some attacks on $E_{\mathtt{KN}}$. Moreover the recents attacks on OCB2[37] by Inoue *et al.* [24, 35, 25, 23] show that it is important to hide tuple $X, E_{\mathtt{KN}}(X)$.

**Corollary 1.** *The knowledge of a pair* $X, E_{\mathtt{KN}}(X)$ *from a direct observation of* $\mathtt{P}_i, \mathtt{C}_i, \mathtt{TAG}, \mathtt{TAG}_{null}$ *is protected by the knowledge of pairs* $\mathtt{PT}_i, \mathtt{CT}_i$*.*

*Proof.* There are two usages of $E_{\mathtt{KN}}$ along the schemes, the ones in the encryption part and the ones in the creation of the TAG. In the encryption part we got $\mathtt{C}_i = \mathtt{CT}_i \oplus E_{\mathtt{KN}}(\mathtt{P}_i \oplus \mathtt{PT}_i)$, thus the knowledge of a pair $(X, E_{\mathtt{KN}}(X))$ is protected by the knowledge of $\mathtt{PT}_i, \mathtt{CT}_i$.

For the authentication part, we have $\mathtt{TAG} = \mathtt{CT}_m \oplus E_{\mathtt{KN}}(\mathtt{PADINFO} \oplus \mathtt{AT}_a \oplus \mathtt{PT}_m \oplus HSWAP(\mathtt{CT}_m))$, thus to know a pair of values $(X, E_{\mathtt{KN}}(X))$ the attacker has to know the couple of $\mathtt{PT}_m, \mathtt{CT}_m$ making the assumption that $\mathtt{AT}_a = \mathtt{AT}_0 = 0$, otherwise the attacker would have to know $E_{\mathtt{K}}(X)$ for some block $X$.

Finally we have $\mathtt{TAG}_{null} = \mathtt{PT}_0 \oplus E_{\mathtt{KN}}(\mathtt{AT}_a \oplus \overline{\mathtt{K}} \oplus \mathtt{PADINFO})$ here again the attacker has to know values which imply to know $\mathtt{PT}_0, \mathtt{CT}_0$ among other values to be able to extract a pair of values $(X, E_{\mathtt{KN}}(X))$. □

**Corollary 2.** *Under the assumptions that the attacker knows all the* $\mathtt{P}_i, \mathtt{C}_i$ *for* $i \in [0, m]$*, respect the* NONCE *uniqueness and format of the inputs. Then the attacker could not predict a computation of* $E_{\mathtt{KN}}(X)$ *on a block* $X$ *before a direct output of* $E_{\mathtt{KN}}$ *in the scheme except if the attacker knows one of the pair* $\mathtt{PT}_i, \mathtt{CT}_i$*.*

*Proof.* Since the change of NONCE affects the key KN in SPAE and the internal variables used in the application of $E_{\mathtt{KN}}$ of SPAE and CSPAE we work with a single NONCE usage. If the attacker is able to make an application of $E_{\mathtt{KN}}$ on same unknown blocks this means that in the encryption part he is able to compute $\mathtt{P}_i, \mathtt{P}_j$ for $i, j \in [0, m]$ such that $\mathtt{P}_i \oplus \mathtt{PT}_i = \mathtt{P}_j \oplus \mathtt{PT}_j$ if we develop this equality, supposing without loss of generality that $j > i$, then we got some equality:

$$\mathtt{PT}_i \oplus \mathtt{P}_i \oplus \mathtt{P}_j = \mathtt{P}_{j-1} \oplus E_{\mathtt{KN}}(\mathtt{P}_{j-1} \oplus E_{\mathtt{KN}}(...\mathtt{P}_i \oplus E_{\mathtt{KN}}(\mathtt{PT}_i \oplus \mathtt{P}_i))).$$

Solving this equality means that the attacker is able to compute successive preimages of $\mathtt{PT}_i$ xored with some data. Therefore such an attack is not possible even with knowing $\mathtt{PT}_i$ since the attacker has not been given knowledge on $E_{\mathtt{KN}}$ (corollary 1).

For the usage of $E_{\mathtt{KN}}$ in producing the TAG there is in addition to the secrecy provided by the unique use of KN the secrecy coming from the internal variables PT, CT (depending of the value of $m$) moreover there are operations such as $HSWAP$ which are exclusive to $\mathtt{TAG}$ computation.

For known block $X$ since for the inputs of the applications of $E_{\mathtt{KN}}$ before a direct output there is always an internal variable (dependant of the $\mathtt{NONCE}$) used it is not possible to determine the inputs on which $E_{\mathtt{KN}}$ is applied.

From proposition 2, knowing a pair $\mathtt{PT}_i, \mathtt{CT}_i$ imply to know every pair of values $\mathtt{PT}_j, \mathtt{CT}_j$ and $\mathtt{KN}$. □

*Remark* 5. If an attacker knows all the $\mathtt{P}_i, \mathtt{C}_i$ for $i \in [0, m]$ and a pair $E_{\mathtt{KN}}(\mathtt{PT}_j \oplus \mathtt{P}_j), \mathtt{PT}_j \oplus \mathtt{P}_j$ then the attacker is able to deduce the value of $\mathtt{KN}$ from $\mathtt{PT}_j, \mathtt{CT}_j$ using proposition 2.

#### 4.1.3 Attack with reuse of outputs

Now we look at some attacks that would replay some output of the scheme (i.e. some $\mathtt{C}_i$) in the inputs (i.e. some $\mathtt{P}_i$) such attack should be taken into account since it could imply some serious security break.

**Proposition 3.** *Under the assumption that the attacker knows all the* $\mathtt{P}_i, \mathtt{C}_i, \mathtt{TAG}, \mathtt{TAG}_{null}$ *the attacker is not able to adapt the values of the* $\mathtt{P}_j, \mathtt{P}_{j+1}$ *to get one of the* $\mathtt{PT}_i$ *or* $\mathtt{CT}_i$*.*

*Proof.* Since it is equivalent to know every couple of $\mathtt{PT}_i, \mathtt{CT}_i$ by proposition 2, we study the ability of an attacker to adapt the values of $\mathtt{P}_0, \mathtt{P}_1$ to get some valuable knowledge on the scheme using also the knowledge of $\mathtt{TAG}$. The proof is a straightforward analysis of the different variables: $\mathtt{TAG}, \mathtt{TAG}_{null}, \mathtt{CT}_0, \mathtt{CT}_1$ and their eventual links with the blocs $\mathtt{P}_i$, with the aim to get in the end a couple of internal variables $\mathtt{PT}_i, \mathtt{CT}_i$. □

#### 4.1.4 Authenticity

In this section we will present bounds for the advantage an adversary has against the scheme. To simplify this analysis we make the assumptions that $\mathtt{PADINFO} = 0$. We first look at all the values used to compute the $\mathtt{TAG}$.

**Proposition 4.** *By the design of the scheme* $\mathtt{AT}_a$ *takes all the possible value.*

*Proof.* Since $E_{\mathtt{K}}$ is supposed to be a permutation then $E_{\mathtt{K}}(\mathtt{AT}_{a-1} \oplus \mathtt{A}_a) = \mathtt{AT}_a$ should take all the possible values. □

Therefore we will make the assumption that there is no additional data since it does not bring restriction to the values taken by the $\mathtt{TAG}$. We study now the values taken by $\mathtt{CT}_m, \mathtt{PT}_m$.

**Lemma 1.** *The values taken by* $\mathtt{CT}_{m+1}$ *and* $\mathtt{PT}_{m+1}$ *are in a set of size lower bounded by the codomain of the function* $x \mapsto x \oplus E_{\mathtt{KN}}(x)$*.*

*Proof.* Since the $\mathtt{PT}_i$ are made recursively by the formula $\mathtt{PT}_i = \mathtt{P}_{i-1} \oplus E_{\mathtt{KN}}(\mathtt{PT}_{i-1} \oplus \mathtt{P}_{i-1})$ we have the values taken by $\mathtt{PT}_i$ which are in a set of size of this codomain. The following values, taking $\mathtt{PT}_{i-1}$ as a constant value, are thus in a set of at least of size of the codomain. Since $\mathtt{CT}_{m+1} = \mathtt{PT}_m \oplus \mathtt{CT}_m$ we get the result. □

We now analyze the possible values for the $\mathtt{TAG} = E_{\mathtt{KN}}(\mathtt{PT}_m \oplus HSWAP(\mathtt{CT}_m) \oplus \mathtt{PADINFO}) \oplus \mathtt{CT}_m$.

**Corollary 3.** *Under the assumption that there is no additional data the values taken by the* $\mathtt{TAG}$ *are lower bounded by the codomain of the function* $x \mapsto x \oplus E_{\mathtt{KN}}(x)$*.*

We remind that we work with the notion of authenticity resilience following [3, Definition 2] and [4].

**Theorem 1.** *If the adversary asks $q$ queries then* $\mathtt{Adv}_{\Pi}^{auth} \leqslant \frac{1}{\Gamma - q}$ *with $\Gamma$ the size of the codomain of the function $x \mapsto x \oplus E_{\mathtt{KN}}(x)$ for* $\mathtt{TAG}$ *and* $\mathtt{Adv}_{\Pi}^{auth} \leqslant \frac{1}{2^{ts}-q}$ *for* $\mathtt{TAG}_{null}$.

*Proof.* We analyze the ability of the adversary to forge a $\mathtt{TAG}$ that will pass the verification. We remind that $\mathtt{TAG} = E_{\mathtt{KN}}(\mathtt{IT} \oplus \mathtt{PADINFO}) \oplus \mathtt{CT}_m$.

Thus even if the attacker knows $\mathtt{IT}$, the attacker needs to know the image of $E_{\mathtt{KN}}$ applied to $\mathtt{IT}$ which is not possible in this setting by the corollary 1. The analysis of the knowledge of $\mathtt{IT}$ has been made in the proof of corollary 1 and is not possible in this context. We conclude with the corollary 3.

For $\mathtt{TAG}_{null} = \mathtt{PT}_0 \oplus E_{\mathtt{KN}}(\mathtt{AT}_a \oplus \overline{\mathtt{K}} \oplus \mathtt{PADINFO})$ since the attacker has not been given any pair of $(X, E_{\mathtt{KN}}(X))$ by corollary 1 then the attacker is not able to compute such a value on a new null message even if the attacker would know $\mathtt{PT}_0$ and $\mathtt{AT}_a \oplus \overline{\mathtt{K}}$.

Thus the attacker has a chance of success of $\frac{1}{\Gamma - q}$ for forging a $\mathtt{TAG}$ and $\frac{1}{2^{ts}-q}$ for forging a $\mathtt{TAG}_{null}$. □

A good resilience of the scheme against $\mathtt{TAG}$ forgery is important to avoid attack in decryption mode. Indeed a decryption of $(\mathtt{NONCE}, C, A, \mathtt{TAG})$ will give substantial informations only if the computed $\mathtt{TAG}$ with the data in input $C, A$ will match the $\mathtt{TAG}$ in input, otherwise the algorithm will output the distinguished value `Failure`.

The only interest to do an attack on the decryption part of the scheme would be to make some $\mathtt{NONCE}$ replay and get some new pair $(P_i, C_i)$ in addition to the ones obtained in encryption mode.

Such an attack need to get at the end a valid $\mathtt{TAG}$. Thus at some point it is needed that some intermediate $\mathtt{PT}_i$ get the same value that the one used to get the original $\mathtt{TAG}$. Thus such an attack reduce to the ones in proof of corollary 2 in term of complexity for the attacker.

An attack like the one in [43] is not possible in this context since any value in the message is always xored with internal values before and after the application of the block encryption function.

#### 4.1.5 Resilience against an attack like the one of Inoue *et al.* [24, 35, 25, 23]

In this subsection we highlight the properties that permit to say that this scheme should be resilient against an attack like the one of [24, 35, 25, 23].

**A good interaction between XEX and XE** As stated in [23] "the vulnerabilities of OCB2 stem from a bad interaction of the XE and XEX components" here in our scheme we always use a XEX (that is to say a Xor Encrypt Xor) on the messages blocks and on the $\mathtt{TAG}$.

**Diffculties to increase knowledge** In the attack of [24, 35, 25, 23] there was the possibility to increase the knowledge of pairs $X, E_{\mathtt{K}}(X)$ through repeated use of the schemes. Here there are two cases to analyse the values $\mathtt{PT}, \mathtt{CT}$ used to mask the encryption function.

- In SPAE they are derived from the function $x \mapsto E_x(x)$ therefore there is only the pair $\mathtt{K}, E_{\mathtt{K}}(\mathtt{K})$ of interest in the scheme and moreover $E_{\mathtt{K}}$ is only used (except $\mathtt{PT}_0, \mathtt{CT}_0$ obviously) for the associated data for which their applications are internal data ($\mathtt{AT}$).
- In CSPAE the use of $E_{\mathtt{K}}$ is done through all the scheme however to know a pair of $\mathtt{PT}, \mathtt{CT}$ associated to a particular $\mathtt{NONCE}$ they need to know the key $\mathtt{K}$ to use the couple $(X \oplus \mathtt{K}, E_{\mathtt{K}}(X))$.

**$\mathtt{TAG}$ forgery** In [24, 35, 25, 23] they manage to forge a $\mathtt{TAG}$ since the operations used for the $\mathtt{TAG}$ were done earlier in the scheme, here $HSWAP$ and $\oplus\overline{\mathtt{K}}$ are used just for the production of the $\mathtt{TAG}$ therefore to produce a valid $\mathtt{TAG}$ the attacker should be able to predict the application of those functions on unknown internal variables. Moreover (in a $\mathtt{NONCE}$ respecting setting) the attacker should not be able to predict a replay of an input of $E_k$ (corollary 2).

### 4.1.6 NONCE misuse

Our scheme is not MRAE secure as in [41, Definition 5]. It fails this definition because of online encryption as stated in [22]. In effect our scheme ciphering $(N, \mathtt{P}_0||...||\mathtt{P}_i||\mathtt{P}_{i+1}||..||\mathtt{P}_m, A)$ and ciphering $(N, \mathtt{P}_0||...||\mathtt{P}_i||\mathtt{P}'_{i+1}||..||\mathtt{P}'_m, A)$ will output for those inputs the same first $\mathtt{C}_0||...||\mathtt{C}_i$ but a different TAG.

**Proposition 5.** *The* NONCE *replay does not permit the attacker to know the input/output of some $E_{\mathtt{KN}}$ in the scheme.*

*Proof.* There are two cases to analyse, the message processing and the production of the TAG. For the message processing by design of the scheme we are not able to get the input/output of $E_{\mathtt{KN}}$ since for the input there is the adding of internal variable $\mathtt{PT}_i$ and for the output there is the adding of the internal variable $\mathtt{CT}_i$, the NONCE replays only permits to repeat those values. The reasoning is similar for the TAG. □

We remind that the NONCE resilience (following [3, 4]) is the ability of the schemes SPAE and CSPAE to do not reveal informations on the scheme with NONCE reuse. Therefore a consequence of proposition 5 is the NONCE resilience of the schemes since the NONCE reuse will not bring information on the key K which is the only relevant information for an analysis on encryption with other NONCE.

*Remark* 6. The adversary can do some differential attack on $E_{\mathtt{K}}$ using NONCE replay with the differentiated input $\mathtt{P}_{i_0}$ observing the resulting differential output $\mathtt{C}_{i_0} = \mathtt{CT}_{i_0} \oplus E_{\mathtt{KN}}(\mathtt{PT}_{i_0} \oplus \mathtt{P}_{i_0})$.

Those differential attacks have some limitations due to the fact that the scheme is a propagating scheme. We could use this differential attack only on one encryption of block, the following operations of the scheme would be very difficult to analyze since they would imply at least the composition of several applications of $E_{\mathtt{KN}}$ on the differentiated inputs.

The same can be done for the TAG by doing input differences on the PADINFO.

By design of the scheme the NONCE replay/misuse does not affect the processing of associated data.

*Remark* 7. The computation of $\mathtt{KN} = \mathtt{K} \oplus \mathtt{NONCE}$ in the design of the *SPAE* scheme forces the user to choose an encryption function $E$ resilient againts related key attacks like the ones of [13, 12]. Therefore, as stated in the work of Biryukov and Nikolic [14] :"this also means that AES-128 is secure against straightforward related-key attacks after 6 rounds", AES-128 is a good choice for block cipher in SPAE.

## 4.2 Differential Fault Analysis

A differential fault analysis ([11]) consists in the production of a pair of output produced from the same inputs and key of $E$ (or $D$) with one of the output produced with a faulty execution the other one in a normal execution. DFA attacks are relevant to smart card and other highly secure chips. In this context it is assumed that the implementation has the means to enforce nonce uniqueness for encryption.

*Remark* 8. This section discuss only DFA, other kind of fault attacks have been published but are not avoided by SPAE and CPAE [16] [17].

### 4.2.1 Associated data processing

Associated data blocks are processed using $E_{\mathtt{K}}$. Both in encryption and decryption, those computation can be done without DFA countermeasures for the simple reason that their outputs, the $\mathtt{AT}_i$ values, are kept as internal variables.

#### 4.2.2 Initialisation

The internal variables $\mathtt{PT}_0$ and $\mathtt{CT}_0$ can be attacked in SPAE, since there is no usage of the NONCE for their computation. The aim of such an attack is to observe those faulty computations directly, but in our scheme those computations are either xored with other data such as $E_{\mathtt{KN}}(\mathtt{P}_0 \oplus \mathtt{PT}_0)$ (or $D_{\mathtt{KN}}(\mathtt{P}_0 \oplus \mathtt{PT}_0)$ in decryption) or processed through $E_{\mathtt{KN}}$ (or $D_{KN}$ in decryption). Therefore it is not necessary (relatively to DFA) to protect the $E_{\mathtt{K}}$ operation producing $\mathtt{PT}_0$ and $\mathtt{CT}_0$.

In CSPAE the internal variables $\mathtt{PT}_0$ and $\mathtt{CT}_0$ cannot be attacked since they use different NONCE for their computation.

#### 4.2.3 SPAE Encryption

Each AEAD encryption use a different NONCE and therefore a different KN. As all outputs are produced using $E_{\mathtt{KN}}$, DFA is not possible using outputs belonging to two different calls to the encryption.

The final $E_{\mathtt{KN}}$ used to compute TAG is a special case: a kind of advanced DFA is possible on it. For example a fault injected on the last round of AES will typically impact only a single byte. The attacker can guess the error pattern by submitting variations of the faulty TAG to the decryption. A successful guess will result in a successful decryption while all other guesses will result in Failure. We nevertheless consider it a likely attack scenario since the guess can be limited to 8 bits. To avoid such attack, an implementation need to protect the final $E_{\mathtt{KN}}$ step used to compute TAG.

To conclude in SPAE encryption only the call to $E$ used for the production of the TAG need DFA countermeasure.

#### 4.2.4 CSPAE Encryption

In CSPAE, $\mathtt{KN} = \mathtt{K}$ so the reasoning done for SPAE does not hold. As all inputs to $E_{\mathtt{KN}}$ are secret values depending on NONCE, the attacker cannot get a faulty output and a fault free output of $E_{\mathtt{KN}}$ with the same input. Within the same AEAD encryption, the attacker cannot attempt a DFA because the attacker is not able to get two $E_{\mathtt{KN}}$ calls with the same input as discussed in corollary 2.

To conclude in CSPAE encryption only the call to $E$ used for the production of the TAG need DFA countermeasure.

#### 4.2.5 SPAE and CSPAE Decryption

In decryption the nonce is in the control of the attacker, as a result the reasoning used for the encryption does not hold. The decryption protects $E_{\mathtt{K}}$, $E_{\mathtt{KN}}$ and $D_{\mathtt{KN}}$ computations nevertheless by leveraging the tag verification.

The approach is to compute TAG from all the outputs of $E_{\mathtt{K}}$, $E_{\mathtt{KN}}$ and $D_{\mathtt{KN}}$ computations. Thanks to this approach, a fault injected in $E_{\mathtt{K}}$, $E_{\mathtt{KN}}$ or $D_{\mathtt{KN}}$ computations always propagates to TAG and therefore the decryption outputs only Failure rather than faulty plain-text.

This approach has some limitations as seen in 4.2.3, 4.2.4 an implementation needs to protect the final $E_{\mathtt{KN}}$ step used to compute TAG.

Now we focus on more sophisticated DFA attacks where the attacker wants to cancel out in the scheme the errors induced before the production of the TAG while getting faulty plaintexts. A fault on the first $D_{\mathtt{KN}}$ propagates to the tag via the $\mathtt{PT}_i$ and $\mathtt{P}_i$, and, depending on the number of blocks, via $\mathtt{CT}_m$. Propagation via $\mathtt{CT}_m$ depends on the number of blocks because the error cancels out every two blocks. For example the error applied on the first computation of $D_{\mathtt{KN}}$ would be in $\mathtt{CT}_2$ but would disappear in $\mathtt{CT}_3$ since it is also present in $\mathtt{PT}_2$. At this point it will still propagate further via $\mathtt{PT}_2$ to $\mathtt{P}_2$ and then $\mathtt{PT}_3$.

This error cancellation phenomenon is the reason for the introduction of the $HSWAP$ operation. Thanks to the $HSWAP$ operation, any non symmetrical error present in both $\mathtt{PT}_m$ and $\mathtt{CT}_m$ does not cancel out and propagates to IT. Even if there is a cancellation of error, there is the xor of $\mathtt{CT}_m$ just before the output of the TAG, in this case the attacker could try to guess the fault by brute force on the faulty value of TAG. The propagation of DFA errors to IT is crucial: it ensures that the resulting faulty TAG cannot be easily guessed. Since IT goes through a call to $E_{\mathtt{KN}}$, even if a single bit is corrupted the resulting TAG will be impractical to guess.

Faults injected on the penultimate $D_{\mathtt{KN}}$ operation propagates only via $\mathtt{PT}_m$. On its way to TAG, the error pattern is xored with PADINFO. An attacker could attempt to manipulate PADINFO in such a way that it would cancel out the error pattern. We mitigate this by formatting PADINFO in a redundant way that it is unable to cancel typical fault patterns on the late AES rounds. Appendix 7.3 gives more details.

To conclude, in SPAE and CSPAE decryption, at most one call to $E$ need DFA countermeasures, all other computations of $E_{\mathtt{K}}$, $E_{\mathtt{KN}}$ and $D_{\mathtt{KN}}$ do not.

## 4.3 Side channel analysis

SPAE and CSPAE do not provide side channel attacks resistance however they are easy to protect when the AES implementation is already protected, which is the typical situation on secure element or secure subsystems. By 'easy to protect' we mean that boolean masking can be used without significantly impacting performances.

## 4.4 Analysis of the privacy

Now that we have studied different possibilities to attack the scheme we study the resilience privacy of the cryptosystem (following [3, Definition 1] and [4]).

First we state a result concerning the number of queries of $E$ we have to take into account for this analysis of the privacy.

*Remark* 9. The relevant number of queries of $E_{\mathtt{K}}/E_{\mathtt{KN}}$ in $SPAE$ and $CSPAE$ for the processing of associated data blocks for an analysis of the privacy is dependant of the equality between $E_{\mathtt{K}}$ and $E_{\mathtt{KN}}$. Therefore, in this analysis we will work with them as different functions when $\mathtt{K} \neq \mathtt{KN}$.

**Lemma 2.** *The relevant number of queries of $E$ in $SPAE$ for the processing of message blocks for an analysis of the privacy is the number of queries with the same* NONCE *since the functions $E_{\mathtt{KN}}, D_{\mathtt{KN}}$ have their key changed with the* NONCE*.*

As mentionned as a consequence of proposition 5 the ability for the attacker to do some NONCE replay except for the one attacked won't give advantages to the attacker however we will mention it's usage in the theorem with the mention "NONCE resilience".

**Theorem 2.** *For an adversary, respecting the* NONCE *resilience, who asks* 1 *(respectively $q$) query $(N^0, A^0, M^0)$ (respectively $(N^i, A^i, M^i)$) that entails $\sigma_e$ (respectively $\sigma_e^c$) blockcipher calls of $E_{\mathtt{KN}}$ on message blocks, $\sigma_a$ blockcipher calls of $E_{\mathtt{K}}$ with non null associated data, then $\mathtt{Adv}^{priv}_{\mathrm{SPAE}} \leqslant \max\left(\frac{1.5(\sigma_e+1)(\sigma_e)}{2^{bs}}, \frac{1.5(\sigma_a)(\sigma_a-1)}{2^{bs}}\right)$ (respectively $\mathtt{Adv}^{priv}_{\mathrm{CSPAE}} \leqslant \frac{1.5(\sigma_e^c+\sigma_a+q)(\sigma_e^c+\sigma_a+q-1)}{2^{bs}}$).*

*Proof.* This proof involves a game playing argument followed by a case-analysis of some collision probabilities as in the proof of [28, lemma 3].

- Game 1: the adversary $\mathcal{A}$ asks the encryption of $q$ queries $(N^i, A^i, M^i)$, the response to each uses of $E_{\mathtt{KN}}$ is stored;

- Game 2: we return a response to each internal query to $E_{\mathtt{KN}}$ by a randomly and uniformly chosen element from $\mathbb{F}_2^n$ with $n = bs$;

- Game 3: we simulate a perfect $2^{bs}$ cipher.

For SPAE there are some separations since $E_{\mathtt{KN}}$ and $E_{\mathtt{K}}$ are different functions. The advantage of $\mathcal{A}$ in distinguishing game 2 and 3 is at most $\max(0.5(\sigma_e+1)(\sigma_e)/2^{bs}, 0.5(\sigma_a)(\sigma_a-1)/2^{bs}))$ for SPAE and $0.5(\sigma_e^c+\sigma_a^c+q)(\sigma_e^c+\sigma_a^c+q-1)/2^{bs}$ for CSPAE. We have thus to focus on the distinction between game 1 and game 2 because we want the attacker to not being able to distinguish game 1 and game 3.

In game 1, we propose to respond to the internal queries (implied by the choice of the adversary) by uniform and random elements of $\mathbb{F}_2^n$. If we have already given an output to another previous input asked then we should sample a new output, a majoration for the probability of this occurring is $\max(0.5(\sigma_e+1)(\sigma_e)/2^{bs}, 0.5(\sigma_a+1)(\sigma_a)/2^{bs})$ for SPAE and $0.5(\sigma_e^c+\sigma_a+q)(\sigma_e^c+\sigma_a+q-1)/2^{bs}$ for CSPAE.

Another bad setting is when we have to produce an output to a previously asked input we must in this case answer according to the previous output produced we are here in a bad setting where we couldn't just sample uniformly as in game 2 a random element of $\mathbb{F}_2^n$.

We have to majorate the probability of this occurrence with the inputs of game 1 which are not directly related to the input of $E_k$. Without loss of generality we suppose that $\mathtt{PADINFO} = 0$. We can not have predictible collisions for the attackers in the inputs of $E_{\mathtt{KN}}, E_{\mathtt{K}}$ by the corollary 2 (we exclude the obvious cases with associated data). There are 2 cases to analyze the ones in the computation of $E_{\mathtt{K}}(\mathtt{P}_i \oplus \mathtt{PT}_i)$, $E_{\mathtt{K}}(\mathtt{AT}_i \oplus \mathtt{A}_i)$ and the computation of $E_{\mathtt{K}}(IT) = E_k(\mathtt{AT}_a \oplus \mathtt{PT}_m \oplus HSWAP(\mathtt{CT}_m))$ which we simplify for the analysis by $E_{\mathtt{K}}(IT) = E_k(\mathtt{AT} \oplus \mathtt{PT} \oplus \mathtt{CT})$.

The first one corresponds to a case where the values $\mathtt{A}_i, \mathtt{P}_i$ are chosen freely. We remind that $\mathtt{PT}_j = E_{\mathtt{KN}}(\mathtt{PT}_{j-1} \oplus \mathtt{P}_{j-1}) \oplus \mathtt{P}_{j-1}$ ($\mathtt{AT}_i = E_{\mathtt{K}}(\mathtt{AT}_{i-1} \oplus \mathtt{A}_{i-1})$) thus their values are determined by the adversary and the choices made by the oracle. We want therefore to majorate the probability that $\mathtt{P}_i = \mathtt{PT}_j \oplus \mathtt{P}_j \oplus \mathtt{PT}_i$ there are $\sigma_e$ such values. We have also to take into account possible equalities with $\mathtt{AT}_i \oplus \mathtt{A}_i$. For SPAE the choices made for $\mathtt{AT}_i$ are taken into account separately. For $CSPAE$ we have to take into account that the equality $\mathtt{P}_i \oplus \mathtt{PT}_i = \mathtt{AT}_j \oplus \mathtt{A}_j$ could occurs therefore we work with a set of size $\sigma_e^c + \sigma_a^c$ values.

The second one (the values $\mathtt{PT}^i, \mathtt{AT}^i, \mathtt{CT}^i$) corresponds to the case where the inputs $\mathtt{PT}, \mathtt{CT}$ values are all results of previously random chosen images of $E_{\mathtt{K}}$. Therefore we majorate the probability of $E_{\mathtt{K}}, E_{\mathtt{KN}}$ called on the same inputs of $\max(\frac{\sigma_e+1(\sigma_e)}{2}, \frac{\sigma_a(\sigma_a-1)}{2}) \times \frac{1}{2^{bs}}$ for SPAE and $\frac{\sigma_e^c+\sigma_a+q(\sigma_e^c+\sigma_a+q-1)}{2} \times \frac{1}{2^{bs}}$ for CSPAE.

The case of the $\mathtt{TAG}_{null}$ has been excluded for obvious reasons. □

**Corollary 4.** *To obtain a level of security of $2^{80}$ with a bs of 128 for SPAE we should limit the usage of a* NONCE *to $2^{32}$ blocks. We obtain a similar result with CSPAE but for the key* K.

Therefore if a lot of data should be used with a single key, more than $2^{39}$ bytes, then it is safer to use SPAE with different NONCE otherwise CSPAE can be used. For example in CAESAR[9] the number of bytes should be greater than $2^{16}$ and for NIST competition on lightweight cryptography [33] it should be greater than $2^{50}-1$.

# 5 Performances

In this section we will present performances of SPAE and compare it to other AE schemes. All implementations are released as an open source project.

## 5.1 Benchmark within MbedTLS on ARM-CortexM4

We inserted SPAE code within the MbedTLS benchmarking project hosted on github.com/wolfeidau/mbedtls. We included three versions:

- SMALL: uses a compact AES implementation, the same used in the ARM-CortexM0 benchmark, see next section.
- FAST: uses the AES from mbedtls, it is a Tbox implementation. This is the implementation which shall be compared against GCM and CCM in this benchmark since they all use the same AES implementation.
- MMCAU: uses the 'MMCAU' accelerator present in K64F MCUs [34]. The SPAE part remains C code, so there is still room for improvement.

This benchmark uses 1024 bytes message without associated data. This confirms that in software implementations on MCUs, SPAE performances are much higher than GCM and CCM. ASCON128 is faster than SPAE in pure software however SPAE can leverage existing AES accelerators to perform even faster.

**Table 2:** MbedTLS benchmark on FRDM-K64F board, 1024 bytes messages

| Algorithm | AES implementation | Kbytes/s | cycles/byte |
|---|---|---|---|
| AES-SPAE-128 | MMCAU | 3101 | 37.8 |
| ASCON128 | - | 1760 | 66.6 |
| AES-SPAE-128 | FAST | 1141 | 102.9 |
| AES-SPAE-128 | SMALL | 546 | 215.1 |
| AES-CCM-128 | FAST | 476 | 246.8 |
| AES-GCM-128 | FAST | 401 | 293.0 |

## 5.2 Benchmark on ARM-CortexM0

We used the cifra library (github.com/ctz/cifra) to get performances of GCM, CCM and OCB mode on ARM-CortexM0 (STM32L011 more precisely). This library uses an AES based on byte oriented 'Sbox' instead of the 32 bit oriented 'Tbox' implementation (like in the CortexM4 benchmark). Due to this the cycles per byte reported here are rather high. Note however that 'Tbox' implementation is rarely used on those devices due to the size of the look up tables: 8KB (STM32L011 has 16KB of flash). All software have been compiled using gcc 7.3.1 and '-Os' optimization level.

**Table 3:** Performance to encrypt and authenticate a 16 bytes message

| | clock cycles | cycles/byte |
|---|---|---|
| SPAE | 18.2K | 1140 |
| CCM | 42.0K | 2627 |
| OCB | 43.0K | 2689 |
| GCM | 65.6K | 4100 |

The results are unusual but make sense once we take into account the following:

- GCM's GHASH is slower in software than AES128 on ARM-CortexM0.
- The benchmark is on a message of 16 bytes, as a result the efficientcy of OCB compared to CCM is not apparent.

# 6 Conclusion and perspectives

We described SPAE and CSPAE, two AEAD algorithms highly suitable for usage in low-cost embedded systems as shown by their performances against other comparables AEAD. We analyzed their security and established proofs for the most common use cases in nonce resilience setting for its security and authenticity.

It would be interesting to see the effective resilience of the scheme against physical attacks and against deep learning based attacks. It is a future work to propose an evaluation for nonce-based authenticated encryption following the security definition in [21, 32, 39].

A follow up work would be to see if SPAE could be used with other block ciphers, SM4[15] for example. This cipher is also hardware accelerated and side channel protected on some secure element. Its security against related key attacks has been analyzed in [45] and therefore it is known to repect remark 7.

The performance benchmark could be completed by including other AEAD based on AES such as SAEB [31].

An interesting work is to take into account in the security analysis the various implementation practices for `NONCE` generation as exposed in [5] and which violate remark 3. The evaluation of the security bounds if a fresh random number is used instead of a true `NONCE` is also something worth investigating since this is a popular practice in real world implementations.

This work is supported by SECURIOT-2-AAP FUI 23, ANR-15-IDEX-02 and partially supported by ANR-15-CE39-0002.

## References

[1] Martin R Albrecht, Kenneth G Paterson, and Gaven J Watson. Plaintext recovery attacks against SSH. In *2009 30th IEEE Symposium on Security and Privacy*, pages 16–26. IEEE, 2009.

[2] Elena Andreeva, Andrey Bogdanov, Nilanjan Datta, Atul Luykx, Bart Mennink, Mridul Nandi, Elmar Tischhauser, and Kan Yasuda. Submission to CAESAR competition: COLM v1, 2015.

[3] Tomer Ashur, Orr Dunkelman, and Atul Luykx. Boosting authenticated encryption robustness with minimal modifications. In *Annual International Cryptology Conference*, pages 3–33. Springer, 2017.

[4] Guy Barwell, Daniel Page, and Martijn Stam. Rogue decryption failures: Reconciling ae robustness notions. In *IMA International Conference on Cryptography and Coding*, pages 94–111. Springer, 2015.

[5] Mihir Bellare, Ruth Ng, and Björn Tackmann. Nonces are noticed: Aead revisited. Cryptology ePrint Archive, Report 2019/624, 2019. https://eprint.iacr.org/2019/624.

[6] Mihir Bellare, Phillip Rogaway, and David Wagner. The EAX Mode of Operation, 2004. https://www.iacr.org/archive/fse2004/30170391/30170391.pdf.

[7] Daniel J. Bernstein. The Poly1305-AES Message-Authentication Code. In *Fast Software Encryption*, pages 32–49. Springer Berlin Heidelberg, 2005.

[8] Daniel J. Bernstein. ChaCha, a variant of Salsa20, 2008.

[9] Daniel J. Bernstein. Caesar: Competition for authenticated encryption: Security, applicability, and robustness, 2014.

[10] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche1, and Ronny Van Keer. Why Keccak is not ARX, 2007.

[11] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 1997.

[12] Alex Biryukov and Dmitry Khovratovich. Related-key cryptanalysis of the full AES-192 and AES-256. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 1–18. Springer, 2009.

[13] Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolić. Distinguisher and related-key attack on the full AES-256. In *Annual International Cryptology Conference*, pages 231–249. Springer, 2009.

[14] Alex Biryukov and Ivica Nikolić. Automatic search for related-key differential characteristics in byte-oriented block ciphers: application to aes, camellia, khazad and others. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 322–344. Springer, 2010.

[15] Whitfield Diffie and George Ledin (translators). Sms4 encryption algorithm for wireless networks. Cryptology ePrint Archive, Report 2008/329, 2008. https://eprint.iacr.org/2008/329.

[16] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Victor Lomné, and Florian Mendel. Statistical fault attacks on nonce-based authenticated encryption schemes. Cryptology ePrint Archive, Report 2016/616, 2016. https://eprint.iacr.org/2016/616.

[17] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. Sifa: Exploiting ineffective fault inductions on symmetric cryptography. Cryptology ePrint Archive, Report 2018/071, 2018. https://eprint.iacr.org/2018/071.

[18] Morris Dworkin. Recommendation for block cipher modes of operation: The CCM mode for authentication and confidentiality. Technical report, National Institute of Standards and Technology, 2004.

[19] Allen Roginsky (NIST) Elaine Barker (NIST). Sp 800-131a rev. 2 - transitioning the use of cryptographic algorithms and key lengths, 2019. https://csrc.nist.gov/publications/detail/sp/800-131a/rev-2/final.

[20] fail0verflow. How to pwn a PS4, 2016. https://fail0verflow.com/media/33c3-slides/#/5.

[21] Shay Gueron and Yehuda Lindell. GCM-SIV: Full nonce misuse-resistant authenticated encryption at under one cycle per byte. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 109–119. ACM, 2015.

[22] Viet Tung Hoang, Reza Reyhanitabar, Phillip Rogaway, and Damian Vizár. Online authenticated-encryption and its nonce-reuse misuse-resistance. In *Annual Cryptology Conference*, pages 493–517. Springer, 2015.

[23] Akiko Inoue, Tetsu Iwata, Kazuhiko Minematsu, and Bertram Poettering. Cryptanalysis of ocb2: Attacks on authenticity and confidentiality. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 3–31, Cham, 2019. Springer International Publishing.

[24] Akiko Inoue and Kazuhiko Minematsu. Cryptanalysis of OCB2. Cryptology ePrint Archive, Report 2018/1040, 2018. https://eprint.iacr.org/2018/1040.

[25] Tetsu Iwata. Plaintext Recovery Attack of OCB2. Cryptology ePrint Archive, Report 2018/1090, 2018. https://eprint.iacr.org/2018/1090.

[26] Tetsu Iwata, Kazuhiko Minematsu, Jian Guo, and Sumio Morioka. Cloc: Authenticated encryption for short input. Cryptology ePrint Archive, Report 2014/157, 2014. https://eprint.iacr.org/2014/157.

[27] Auguste Kerckhoffs. "La cryptographie militaire" Journal des sciences militaires, 1883. vol. IX, pp. 5–83, January 1883, pp. 161–191, February 1883.

[28] Ted Krovetz and Phillip Rogaway. The software performance of authenticated-encryption modes. In *International Workshop on Fast Software Encryption*, pages 306–327. Springer, 2011.

[29] Moses Liskov, Ronald L Rivest, and David Wagner. Tweakable block ciphers. *Journal of cryptology*, 24(3):588–613, 2011.

[30] David A. McGrew and John Viega. The security and performance of the galois/counter mode of operation (full version). Cryptology ePrint Archive, Report 2004/193, 2004. https://eprint.iacr.org/2004/193.

[31] Yusuke Naito, Mitsuru Matsui, Takeshi Sugawara, and Daisuke Suzuki. Saeb: A lightweight blockcipher-based aead mode of operation. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 192–217, 2018.

[32] Chanathip Namprempre, Phillip Rogaway, and Thomas Shrimpton. Reconsidering generic composition. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 257–274. Springer, 2014.

[33] NIST. NIST Lightweight Cryptography Standardization Process, 2019.

[34] NXP. K64 Sub-Family Reference Manual, 2017.

[35] Bertram Poettering. Breaking the confidentiality of OCB2. Cryptology ePrint Archive, Report 2018/1087, 2018. `https://eprint.iacr.org/2018/1087`.

[36] Phillip Rogaway. Authenticated-encryption with associated-data. In *Proceedings of the 9th Annual Conference on Computer and Communications Security (CCS-9)*, pages 98–107. ACM, 2002.

[37] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 16–31. Springer, 2004.

[38] Phillip Rogaway, Mihir Bellare, and John Black. OCB: A Block-cipher Mode of Operation for Efficient Authenticated Encryption. *ACM Trans. Inf. Syst. Secur.*, 6(3):365–403, aug 2003.

[39] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 373–390. Springer, 2006.

[40] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006*, pages 373–390, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[41] Phillip Rogaway and Thomas Shrimpton. The SIV Mode of Operation for Deterministic Authenticated-Encryption (Key Wrap) and Misuse-Resistant Nonce-Based Authenticated-Encryption, 2007.

[42] Phillip Rogaway and David Wagner. A Critique of CCM, 2003. `http://web.cs.ucdavis.edu/~rogaway/papers/ccm.pdf`.

[43] Serge Vaudenay. Security Flaws Induced by CBC Padding—Applications to SSL, IPSEC, WTLS... In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 534–545. Springer, 2002.

[44] Damian Vizár. The state of the authenticated encryption. *Tatra Mountains Mathematical Publications*, 67:167–190, 09 2016.

[45] Jian Zhang, Wenling Wu, and Yafei Zheng. Security of sm4 against (related-key) differential cryptanalysis. In *Information Security Practice and Experience*, pages 65–78, 2016.

# 7 Appendix

## 7.1 SPAE-AES128 test vectors

```
m=0,a=0
SPAE - encryption
key                =  b'00000000000000000000000000000001'
nonce              =  b'00000000000000000000000000000002'
message            =
associated data    =
               PT0 =  b'a17e9f69e4f25a8b8620b4af78eefd6e'
               CT0 =  b'a17e9f69e4f25a8b8620b4af78eefd6f'
                MT =  b'fffffffffffffffffffffffffffffffe'
                IT =  b'fffffffffffffffffffffffffffffffe'
           PADINFO =  b'00000000000000000000000000000000'
authentication tag =  b'6b52a86d2741165af5ad9b4694d978e7'
out                =  b'6b52a86d2741165af5ad9b4694d978e7'
```

```
m=0,a=1
SPAE - encryption
key                =  b'00000000000000000000000000000001'
nonce              =  b'00000000000000000000000000000002'
message            =
associated data    =  b'00000000000000000000000000000006'
               PT0 =  b'a17e9f69e4f25a8b8620b4af78eefd6e'
               CT0 =  b'a17e9f69e4f25a8b8620b4af78eefd6f'
               AT1 =  b'131527259dc7975992e4bbfd0510e9fa'
                MT =  b'fffffffffffffffffffffffffffffffe'
                IT =  b'ecead8da623868a66d1b4402faef1604'
           PADINFO =  b'00000000800000000000000080000000'
authentication tag =  b'840fa2e1542e22a1146b8ccb4f98410f'
out                =  b'840fa2e1542e22a1146b8ccb4f98410f'
```

```
m=1,a=0
SPAE - encryption
key                =  b'00000000000000000000000000000001'
nonce              =  b'00000000000000000000000000000002'
message            =  b'00000000000000000000000000000003'
associated data    =
               PT0 =  b'a17e9f69e4f25a8b8620b4af78eefd6e'
               CT0 =  b'a17e9f69e4f25a8b8620b4af78eefd6f'
               P0  =  b'00000000000000000000000000000003'
               C0  =  b'731bdd384f415c11081d08ecdc3efe5d'
               PT1 =  b'd2654251abb3069a8e3dbc43a4d00331'
               CT1 =  b'00000000000000000000000000000001'
                MT =  b'd2654251abb3069b8e3dbc43a4d00331'
                IT =  b'd2654251abb3069b8e3dbc43a4d00331'
           PADINFO =  b'80000000000000008000000000000000'
authentication tag =  b'8f11c2f7f934270ebbd7c3033fbbabef'
out                =  b'731bdd384f415c11081d08ecdc3efe5d8f11c2f7f934270
    ↪ ebbd7c3033fbbabef'
```

```
m=2,a=0
SPAE - encryption
key                = b'00000000000000000000000000000001'
nonce              = b'00000000000000000000000000000002'
message            = b'000000000000000000000000000000030000000000000000
    ↪ 0000000000000004'
associated data    =
               PT0 = b'a17e9f69e4f25a8b8620b4af78eefd6e'
               CT0 = b'a17e9f69e4f25a8b8620b4af78eefd6f'
               P0  = b'00000000000000000000000000000003'
               C0  = b'731bdd384f415c11081d08ecdc3efe5d'
               PT1 = b'd2654251abb3069a8e3dbc43a4d00331'
               CT1 = b'00000000000000000000000000000001'
               P1  = b'00000000000000000000000000000004'
               C1  = b'd454792a75871ce616511d13983f9681'
               PT2 = b'd454792a75871ce616511d13983f9684'
               CT2 = b'd2654251abb3069a8e3dbc43a4d00330'
                MT = b'5a69c569d1571fd6c4345f42338c901e'
                IT = b'5a69c569d1571fd6c4345f42338c901e'
           PADINFO = b'00010000000000000001000000000000'
authentication tag = b'773ff95c3282ff9ea8794295685191ea'
out                = b'731bdd384f415c11081d08ecdc3efe5dd454792a75871ce61
    ↪ 6511d13983f9681773ff95c3282ff9ea8794295685191ea'
```

```
m=3,a=0
SPAE - encryption
key                = b'00000000000000000000000000000001'
nonce              = b'00000000000000000000000000000002'
message            = b'000000000000000000000000000000030000000000000000
    ↪ 00000000000000040000000000000000000000000000000005'
associated data    =
               PT0 = b'a17e9f69e4f25a8b8620b4af78eefd6e'
               CT0 = b'a17e9f69e4f25a8b8620b4af78eefd6f'
               P0  = b'00000000000000000000000000000003'
               C0  = b'731bdd384f415c11081d08ecdc3efe5d'
               PT1 = b'd2654251abb3069a8e3dbc43a4d00331'
               CT1 = b'00000000000000000000000000000001'
               P1  = b'00000000000000000000000000000004'
               C1  = b'd454792a75871ce616511d13983f9681'
               PT2 = b'd454792a75871ce616511d13983f9684'
               CT2 = b'd2654251abb3069a8e3dbc43a4d00330'
               P2  = b'00000000000000000000000000000005'
               C2  = b'406d307c0f1f9a95878e7bb968108aaa'
               PT3 = b'9208722da4ac9c0f09b3c7faccc0899f'
               CT3 = b'06313b7bde341a7c986ca1503cef95b4'
                MT = b'0a64d37d984309bb0f82fc8112f493e3'
                IT = b'0a64d37d984309bb0f82fc8112f493e3'
           PADINFO = b'80010000000000008001000000000000'
authentication tag = b'a4d864382672b6abbfeb80563bbfefa1'
out                = b'731bdd384f415c11081d08ecdc3efe5dd454792a75871ce61
    ↪ 6511d13983f9681406d307c0f1f9a95878e7bb968108aaaa4d864382672b6abbfeb
    ↪ 80563bbfefa1'
```

```
m=3,a=1
SPAE - encryption
key              =  b'00000000000000000000000000000001'
nonce            =  b'00000000000000000000000000000002'
message          =  b'0000000000000000000000000000000300000000000000000
    ↪ 0000000000000400000000000000000000000000000005'
associated data  =  b'00000000000000000000000000000006'
              PT0 =  b'a17e9f69e4f25a8b8620b4af78eefd6e'
              CT0 =  b'a17e9f69e4f25a8b8620b4af78eefd6f'
              P0  =  b'00000000000000000000000000000003'
              C0  =  b'731bdd384f415c11081d08ecdc3efe5d'
              PT1 =  b'd2654251abb3069a8e3dbc43a4d00331'
              CT1 =  b'00000000000000000000000000000001'
              P1  =  b'00000000000000000000000000000004'
              C1  =  b'd454792a75871ce616511d13983f9681'
              PT2 =  b'd454792a75871ce616511d13983f9684'
              CT2 =  b'd2654251abb3069a8e3dbc43a4d00330'
              P2  =  b'00000000000000000000000000000005'
              C2  =  b'406d307c0f1f9a95878e7bb968108aaa'
              PT3 =  b'9208722da4ac9c0f09b3c7faccc0899f'
              CT3 =  b'06313b7bde341a7c986ca1503cef95b4'
              AT1 =  b'131527259dc7975992e4bbfd0510e9fa'
               MT =  b'0a64d37d984309bb0f82fc8112f493e3'
               IT =  b'1971f45805849ee29d66477c17e47a19'
          PADINFO =  b'80010000800000008001000080000000'
authentication tag = b'b2d2286e176bbe8120af02dd378a22f0'
out              =  b'731bdd384f415c11081d08ecdc3efe5dd454792a75871ce61
    ↪ 6511d13983f9681406d307c0f1f9a95878e7bb968108aaab2d2286e176bbe8120af
    ↪ 02dd378a22f0'
```

```
m=3,a=2
SPAE - encryption
key              =  b'00000000000000000000000000000001'
nonce            =  b'00000000000000000000000000000002'
message          =  b'0000000000000000000000000000000300000000000000000
    ↪ 0000000000000400000000000000000000000000000005'
associated data  =  b'0000000000000000000000000000000600000000000000000
    ↪ 000000000000007'
              PT0 =  b'a17e9f69e4f25a8b8620b4af78eefd6e'
              CT0 =  b'a17e9f69e4f25a8b8620b4af78eefd6f'
              P0  =  b'00000000000000000000000000000003'
              C0  =  b'731bdd384f415c11081d08ecdc3efe5d'
              PT1 =  b'd2654251abb3069a8e3dbc43a4d00331'
              CT1 =  b'00000000000000000000000000000001'
              P1  =  b'00000000000000000000000000000004'
              C1  =  b'd454792a75871ce616511d13983f9681'
              PT2 =  b'd454792a75871ce616511d13983f9684'
              CT2 =  b'd2654251abb3069a8e3dbc43a4d00330'
              P2  =  b'00000000000000000000000000000005'
              C2  =  b'406d307c0f1f9a95878e7bb968108aaa'
              PT3 =  b'9208722da4ac9c0f09b3c7faccc0899f'
```

```
              CT3 =  b'06313b7bde341a7c986ca1503cef95b4'
              AT1 =  b'131527259dc7975992e4bbfd0510e9fa'
              AT2 =  b'dba2c410fd3d0a66d02984fc6e85d61a'
               MT =  b'0a64d37d984309bb0f82fc8112f493e3'
               IT =  b'd1c6176d657e03dddfab787d7c7145f9'
          PADINFO =  b'80010000000100008001000000010000'
authentication tag =  b'baf2944c6cf3b3a0883a024b23f34fec'
out                =  b'731bdd384f415c11081d08ecdc3efe5dd454792a75871ce61
    ↪ 6511d13983f9681406d307c0f1f9a95878e7bb968108aaabaf2944c6cf3b3a0883a
    ↪ 024b23f34fec'
```

```
m=3,a=3
SPAE - encryption
key                =  b'00000000000000000000000000000001'
nonce              =  b'00000000000000000000000000000002'
message            =  b'000000000000000000000000000000030000000000000000
    ↪ 00000000000000040000000000000000000000000000000005'
associated data    =  b'000000000000000000000000000000060000000000000000
    ↪ 00000000000000070000000000000000000000000000000008'
              PT0 =  b'a17e9f69e4f25a8b8620b4af78eefd6e'
              CT0 =  b'a17e9f69e4f25a8b8620b4af78eefd6f'
              P0  =  b'00000000000000000000000000000003'
              C0  =  b'731bdd384f415c11081d08ecdc3efe5d'
              PT1 =  b'd2654251abb3069a8e3dbc43a4d00331'
              CT1 =  b'00000000000000000000000000000001'
              P1  =  b'00000000000000000000000000000004'
              C1  =  b'd454792a75871ce616511d13983f9681'
              PT2 =  b'd454792a75871ce616511d13983f9684'
              CT2 =  b'd2654251abb3069a8e3dbc43a4d00330'
              P2  =  b'00000000000000000000000000000005'
              C2  =  b'406d307c0f1f9a95878e7bb968108aaa'
              PT3 =  b'9208722da4ac9c0f09b3c7faccc0899f'
              CT3 =  b'06313b7bde341a7c986ca1503cef95b4'
              AT1 =  b'131527259dc7975992e4bbfd0510e9fa'
              AT2 =  b'dba2c410fd3d0a66d02984fc6e85d61a'
              AT3 =  b'7152d55728d8c03e547ab64cd21531fa'
               MT =  b'0a64d37d984309bb0f82fc8112f493e3'
               IT =  b'7b36062ab09bc9855bf84acdc0e1a219'
          PADINFO =  b'80010000800100008001000080010000'
authentication tag =  b'6606f31a266516b3f3c57529ef402421'
out                =  b'731bdd384f415c11081d08ecdc3efe5dd454792a75871ce61
    ↪ 6511d13983f9681406d307c0f1f9a95878e7bb968108aaa6606f31a266516b3f3c5
    ↪ 7529ef402421'
SPAE - decryption
key                =  b'00000000000000000000000000000001'
nonce              =  b'00000000000000000000000000000002'
message            =  b'731bdd384f415c11081d08ecdc3efe5dd454792a75871ce61
    ↪ 6511d13983f9681406d307c0f1f9a95878e7bb968108aaa'
associated data    =  b'000000000000000000000000000000060000000000000000
    ↪ 00000000000000070000000000000000000000000000000008'
              PT0 =  b'a17e9f69e4f25a8b8620b4af78eefd6e'
              CT0 =  b'a17e9f69e4f25a8b8620b4af78eefd6f'
```

```
                P0   =  b'00000000000000000000000000000003'
                C0   =  b'731bdd384f415c11081d08ecdc3efe5d'
                PT1  =  b'd2654251abb3069a8e3dbc43a4d00331'
                CT1  =  b'00000000000000000000000000000001'
                P1   =  b'00000000000000000000000000000004'
                C1   =  b'd454792a75871ce616511d13983f9681'
                PT2  =  b'd454792a75871ce616511d13983f9684'
                CT2  =  b'd2654251abb3069a8e3dbc43a4d00330'
                P2   =  b'00000000000000000000000000000005'
                C2   =  b'406d307c0f1f9a95878e7bb968108aaa'
                PT3  =  b'9208722da4ac9c0f09b3c7faccc0899f'
                CT3  =  b'06313b7bde341a7c986ca1503cef95b4'
                AT1  =  b'131527259dc7975992e4bbfd0510e9fa'
                AT2  =  b'dba2c410fd3d0a66d02984fc6e85d61a'
                AT3  =  b'7152d55728d8c03e547ab64cd21531fa'
                 MT  =  b'0a64d37d984309bb0f82fc8112f493e3'
                 IT  =  b'7b36062ab09bc9855bf84acdc0e1a219'
            PADINFO  =  b'80010000800100008001000080010000'
authentication tag  =  b'6606f31a266516b3f3c57529ef402421'
provided tag        =  b'6606f31a266516b3f3c57529ef402421'
out                 =  b'000000000000000000000000000000030000000000000000
    ↪ 0000000000000004000000000000000000000000000000005'
```

```
m=3,a=3 padded
SPAE - encryption
key                 =  b'00000000000000000000000000000001'
nonce               =  b'00000000000000000000000000000002'
message             =  b'000000000000000000000000000000030000000000000000
    ↪ 0000000000000409'
associated data     =  b'000000000000000000000000000000060000000000000000
    ↪ 0000000000000070a0b'
                PT0  =  b'a17e9f69e4f25a8b8620b4af78eefd6e'
                CT0  =  b'a17e9f69e4f25a8b8620b4af78eefd6f'
                P0   =  b'00000000000000000000000000000003'
                C0   =  b'731bdd384f415c11081d08ecdc3efe5d'
                PT1  =  b'd2654251abb3069a8e3dbc43a4d00331'
                CT1  =  b'00000000000000000000000000000001'
                P1   =  b'00000000000000000000000000000004'
                C1   =  b'd454792a75871ce616511d13983f9681'
                PT2  =  b'd454792a75871ce616511d13983f9684'
                CT2  =  b'd2654251abb3069a8e3dbc43a4d00330'
                P2   =  b'09000000000000000000000000000000'
                C2   =  b'804fcc83143603242c36fe10cab4de85'
                PT3  =  b'5b2a8ed2bf8505bea20b42536e64ddb5'
                CT3  =  b'06313b7bde341a7c986ca1503cef95b4'
                AT1  =  b'131527259dc7975992e4bbfd0510e9fa'
                AT2  =  b'dba2c410fd3d0a66d02984fc6e85d61a'
                AT3  =  b'd23963e8cabeaa4a76899b5f5e3ee58d'
                 MT  =  b'c3462f82836a900aa43a7928b050c7c9'
                 IT  =  b'117f4c6a49d43a40d2b3e277ee6e2244'
            PADINFO  =  b'08010000100100000801000010010000'
authentication tag  =  b'5c2209f570ef626cb211725de2a9af06'
```

```
out                  =  b'731bdd384f415c11081d08ecdc3efe5dd454792a75871ce61
    ↪ 6511d13983f9681804fcc83143603242c36fe10cab4de855c2209f570ef626cb211
    ↪ 725de2a9af06'
SPAE - decryption
key                  =  b'00000000000000000000000000000001'
nonce                =  b'00000000000000000000000000000002'
message              =  b'731bdd384f415c11081d08ecdc3efe5dd454792a75871ce61
    ↪ 6511d13983f9681804fcc83143603242c36fe10cab4de85'
associated data      =  b'0000000000000000000000000000000600000000000000000
    ↪ 0000000000000070a0b'
                  PT0 =  b'a17e9f69e4f25a8b8620b4af78eefd6e'
                  CT0 =  b'a17e9f69e4f25a8b8620b4af78eefd6f'
                  P0  =  b'00000000000000000000000000000003'
                  C0  =  b'731bdd384f415c11081d08ecdc3efe5d'
                  PT1 =  b'd2654251abb3069a8e3dbc43a4d00331'
                  CT1 =  b'00000000000000000000000000000001'
                  P1  =  b'00000000000000000000000000000004'
                  C1  =  b'd454792a75871ce616511d13983f9681'
                  PT2 =  b'd454792a75871ce616511d13983f9684'
                  CT2 =  b'd2654251abb3069a8e3dbc43a4d00330'
                  P2  =  b'09000000000000000000000000000000'
                  C2  =  b'804fcc83143603242c36fe10cab4de85'
                  PT3 =  b'5b2a8ed2bf8505bea20b42536e64ddb5'
                  CT3 =  b'06313b7bde341a7c986ca1503cef95b4'
                  AT1 =  b'131527259dc7975992e4bbfd0510e9fa'
                  AT2 =  b'dba2c410fd3d0a66d02984fc6e85d61a'
                  AT3 =  b'd23963e8cabeaa4a76899b5f5e3ee58d'
                   MT =  b'c3462f82836a900aa43a7928b050c7c9'
                   IT =  b'117f4c6a49d43a40d2b3e277ee6e2244'
              PADINFO =  b'08010000100100000801000010010000'
authentication tag =  b'5c2209f570ef626cb211725de2a9af06'
provided tag         =  b'5c2209f570ef626cb211725de2a9af06'
out                  =  b'0000000000000000000000000000000300000000000000000
    ↪ 00000000000000409'
```

```
SPAE - encryption
key                  =  b'000102030405060708090a0b0c0d0e0f'
nonce                =  b'000102030405060708090a0b0c0d0e0f'
message              =
associated data      =
                  PT0 =  b'0a9509b6456bf642f9ca9e53ca5ee455'
                  CT0 =  b'0a940bb5416ef045f1c39458c653ea5a'
                   MT =  b'fffefdfcfbfaf9f8f7f6f5f4f3f2f1f0'
                   IT =  b'fffefdfcfbfaf9f8f7f6f5f4f3f2f1f0'
              PADINFO =  b'00000000000000000000000000000000'
authentication tag =  b'0a52cf639cf84370fe50b76d60eff179'
out                  =  b'0a52cf639cf84370fe50b76d60eff179'
```

```
SPAE - encryption
key                  =  b'000102030405060708090a0b0c0d0e0f'
nonce                =  b'000102030405060708090a0b0c0d0e0f'
message              =  b'000102030405060708090a0b0c0d0e0f'
```

```
associated data    =  b'000102030405060708090a0b0c0d0e0f'
               PT0 =  b'0a9509b6456bf642f9ca9e53ca5ee455'
               CT0 =  b'0a940bb5416ef045f1c39458c653ea5a'
               P0  =  b'000102030405060708090a0b0c0d0e0f'
               C0  =  b'9f7562a92c45ee0719ef6b6586554360'
               PT1 =  b'95e06b1f692e1845e025f5364c0ba735'
               CT1 =  b'000102030405060708090a0b0c0d0e0f'
               AT1 =  b'0a940bb5416ef045f1c39458c653ea5a'
                MT =  b'9de961146523164ae024f735480ea132'
                IT =  b'977d6aa1244de60f11e7636d8e5d4b68'
           PADINFO =  b'80000000800000008000000080000000'
authentication tag =  b'b524324d75cef37f1f2bc1ad2b242db8'
out                =  b'9f7562a92c45ee0719ef6b6586554360b524324d75cef37f1
    ↪ f2bc1ad2b242db8'
```

```
SPAE - encryption
key                =  b'000102030405060708090a0b0c0d0e0f'
nonce              =  b'000102030405060708090a0b0c0d0e0f'
message            =  b'000102030405060708090a0b0c0d0e0f10111213141516171
    ↪ 8191a1b1c1d1e1f'
associated data    =  b'000102030405060708090a0b0c0d0e0f10111213141516171
    ↪ 8191a1b1c1d1e'
               PT0 =  b'0a9509b6456bf642f9ca9e53ca5ee455'
               CT0 =  b'0a940bb5416ef045f1c39458c653ea5a'
               P0  =  b'000102030405060708090a0b0c0d0e0f'
               C0  =  b'9f7562a92c45ee0719ef6b6586554360'
               PT1 =  b'95e06b1f692e1845e025f5364c0ba735'
               CT1 =  b'000102030405060708090a0b0c0d0e0f'
               P1  =  b'101112131415161718191a1b1c1d1e1f'
               C1  =  b'80df406383afdf4ef689443e2c82916b'
               PT2 =  b'90cf507393bfcf5ee699542e3c92817b'
               CT2 =  b'95e1691c6d2b1e42e82cff3d4006a93a'
               AT1 =  b'0a940bb5416ef045f1c39458c653ea5a'
               AT2 =  b'68e2c19fa9096698ae35d29b54b0c601'
                MT =  b'78e3af4ed3b9666473783d3251b99f39'
                IT =  b'10016ed17ab000fcdd4defa905095938'
           PADINFO =  b'00010000f800000000010000f8000000'
authentication tag =  b'60dc7498e5e41a0ad07bd975ed5e97a3'
out                =  b'9f7562a92c45ee0719ef6b658655436080df406383afdf4ef
    ↪ 689443e2c82916b60dc7498e5e41a0ad07bd975ed5e97a3'
```

```
SPAE - encryption
key                =  b'000102030405060708090a0b0c0d0e0f'
nonce              =  b'000102030405060708090a0b0c0d0e0f'
message            =  b'000102030405060708090a0b0c0d0e0f10111213141516171
    ↪ 8191a1b1c1d1e1f'
associated data    =  b'000102030405060708090a0b0c0d0e0f10111213141516171
    ↪ 8191a1b1c1d1e1f'
               PT0 =  b'0a9509b6456bf642f9ca9e53ca5ee455'
               CT0 =  b'0a940bb5416ef045f1c39458c653ea5a'
               P0  =  b'000102030405060708090a0b0c0d0e0f'
               C0  =  b'9f7562a92c45ee0719ef6b6586554360'
```

```
             PT1 =  b'95e06b1f692e1845e025f5364c0ba735'
             CT1 =  b'000102030405060708090a0b0c0d0e0f'
             P1  =  b'101112131415161718191a1b1c1d1e1f'
             C1  =  b'80df406383afdf4ef689443e2c82916b'
             PT2 =  b'90cf507393bfcf5ee699542e3c92817b'
             CT2 =  b'95e1691c6d2b1e42e82cff3d4006a93a'
             AT1 =  b'0a940bb5416ef045f1c39458c653ea5a'
             AT2 =  b'3cf456b4ca488aa383c79c98b34797cb'
              MT =  b'78e3af4ed3b9666473783d3251b99f39'
              IT =  b'4417f9fa19f1ecc7f0bfa1aae2fe08f2'
         PADINFO =  b'00010000000100000001000000010000'
authentication tag =  b'697844f03d7e73f226d888d556f53058'
out                =  b'9f7562a92c45ee0719ef6b658655436080df406383afdf4ef
    ↪ 689443e2c82916b697844f03d7e73f226d888d556f53058'
```

## 7.2 CSPAE-AES128 test vectors

```
m=0,a=0
CSPAE - encryption
key                =  b'00000000000000000000000000000001'
nonce              =  b'00000000000000000000000000000002'
message            =
associated data    =
             PT0 =  b'd0017f493d6d576c9ea7d5683209ff1b'
             CT0 =  b'd0017f493d6d576c9ea7d5683209ff18'
              MT =  b'fffffffffffffffffffffffffffffffe'
              IT =  b'fffffffffffffffffffffffffffffffe'
         PADINFO =  b'00000000000000000000000000000000'
authentication tag =  b'0bec7271c5d3f69c28d934da38f0ac8c'
out                =  b'0bec7271c5d3f69c28d934da38f0ac8c'
```

```
m=0,a=1
CSPAE - encryption
key                =  b'00000000000000000000000000000001'
nonce              =  b'00000000000000000000000000000002'
message            =
associated data    =  b'00000000000000000000000000000006'
             PT0 =  b'd0017f493d6d576c9ea7d5683209ff1b'
             CT0 =  b'd0017f493d6d576c9ea7d5683209ff18'
             AT1 =  b'131527259dc7975992e4bbfd0510e9fa'
              MT =  b'fffffffffffffffffffffffffffffffe'
              IT =  b'ecead8da623868a66d1b4402faef1604'
         PADINFO =  b'00000000800000000000000080000000'
authentication tag =  b'74600b9d86873ce2999a6928ed9ac152'
out                =  b'74600b9d86873ce2999a6928ed9ac152'
```

```
m=1,a=0
CSPAE - encryption
key                =  b'00000000000000000000000000000001'
nonce              =  b'00000000000000000000000000000002'
message            =  b'00000000000000000000000000000003'
```

```
associated data    =
               PT0 =  b'd0017f493d6d576c9ea7d5683209ff1b'
               CT0 =  b'd0017f493d6d576c9ea7d5683209ff18'
               P0  =  b'00000000000000000000000000000003'
               C0  =  b'af06863bfe5ab6f4d07ef32afba1baea'
               PT1 =  b'7f07f972c337e1984ed92642c9a845f1'
               CT1 =  b'00000000000000000000000000000003'
                MT =  b'7f07f972c337e19b4ed92642c9a845f1'
                IT =  b'7f07f972c337e19b4ed92642c9a845f1'
           PADINFO =  b'80000000000000008000000000000000'
authentication tag =  b'69d6f0bbc6c56a135b4cb34b6752c7bd'
out                =  b'af06863bfe5ab6f4d07ef32afba1baea69d6f0bbc6c56a135
    ↪ b4cb34b6752c7bd'
```

```
m=2,a=0
CSPAE - encryption
key                =  b'00000000000000000000000000000001'
nonce              =  b'00000000000000000000000000000002'
message            =  b'0000000000000000000000000000000300000000000000000
    ↪ 000000000000004'
associated data    =
               PT0 =  b'd0017f493d6d576c9ea7d5683209ff1b'
               CT0 =  b'd0017f493d6d576c9ea7d5683209ff18'
               P0  =  b'00000000000000000000000000000003'
               C0  =  b'af06863bfe5ab6f4d07ef32afba1baea'
               PT1 =  b'7f07f972c337e1984ed92642c9a845f1'
               CT1 =  b'00000000000000000000000000000003'
               P1  =  b'00000000000000000000000000000004'
               C1  =  b'ecd2adc6b87c84f9a9f079b100f5bc96'
               PT2 =  b'ecd2adc6b87c84f9a9f079b100f5bc91'
               CT2 =  b'7f07f972c337e1984ed92642c9a845f2'
                MT =  b'a20b8b8471d4c10bd6f780c3c3c25d09'
                IT =  b'a20b8b8471d4c10bd6f780c3c3c25d09'
           PADINFO =  b'00010000000000000001000000000000'
authentication tag =  b'de39ac5f602ef05afc8729933de7b8be'
out                =  b'af06863bfe5ab6f4d07ef32afba1baeaecd2adc6b87c84f9a
    ↪ 9f079b100f5bc96de39ac5f602ef05afc8729933de7b8be'
```

```
m=3,a=0
CSPAE - encryption
key                =  b'00000000000000000000000000000001'
nonce              =  b'00000000000000000000000000000002'
message            =  b'0000000000000000000000000000000300000000000000000
    ↪ 00000000000000400000000000000000000000000000005'
associated data    =
               PT0 =  b'd0017f493d6d576c9ea7d5683209ff1b'
               CT0 =  b'd0017f493d6d576c9ea7d5683209ff18'
               P0  =  b'00000000000000000000000000000003'
               C0  =  b'af06863bfe5ab6f4d07ef32afba1baea'
               PT1 =  b'7f07f972c337e1984ed92642c9a845f1'
               CT1 =  b'00000000000000000000000000000003'
               P1  =  b'00000000000000000000000000000004'
```

```
                 C1   =  b'ecd2adc6b87c84f9a9f079b100f5bc96'
                 PT2  =  b'ecd2adc6b87c84f9a9f079b100f5bc91'
                 CT2  =  b'7f07f972c337e1984ed92642c9a845f2'
                 P2   =  b'00000000000000000000000000000005'
                 C2   =  b'38d4e578462b696ca7aed596e3fd14e3'
                 PT3  =  b'47d31c0a851c88f4e977f3d42a555114'
                 CT3  =  b'93d554b47b4b6561e7295ff3c95df963'
                  MT  =  b'a0fa43f94c4171977aa2a760511e3475'
                  IT  =  b'a0fa43f94c4171977aa2a760511e3475'
             PADINFO  =  b'80010000000000008001000000000000'
authentication tag =  b'ddbd5c3f4573463da81445b8cc221bea'
out                   =  b'af06863bfe5ab6f4d07ef32afba1baeaecd2adc6b87c84f9a
    ↪ 9f079b100f5bc9638d4e578462b696ca7aed596e3fd14e3ddbd5c3f4573463da814
    ↪ 45b8cc221bea'
```

```
m=3,a=1
CSPAE - encryption
key                   =  b'00000000000000000000000000000001'
nonce                 =  b'00000000000000000000000000000002'
message               =  b'000000000000000000000000000000030000000000000000
    ↪ 000000000000000400000000000000000000000000000005'
associated data       =  b'00000000000000000000000000000006'
                 PT0  =  b'd0017f493d6d576c9ea7d5683209ff1b'
                 CT0  =  b'd0017f493d6d576c9ea7d5683209ff18'
                 P0   =  b'00000000000000000000000000000003'
                 C0   =  b'af06863bfe5ab6f4d07ef32afba1baea'
                 PT1  =  b'7f07f972c337e1984ed92642c9a845f1'
                 CT1  =  b'00000000000000000000000000000003'
                 P1   =  b'00000000000000000000000000000004'
                 C1   =  b'ecd2adc6b87c84f9a9f079b100f5bc96'
                 PT2  =  b'ecd2adc6b87c84f9a9f079b100f5bc91'
                 CT2  =  b'7f07f972c337e1984ed92642c9a845f2'
                 P2   =  b'00000000000000000000000000000005'
                 C2   =  b'38d4e578462b696ca7aed596e3fd14e3'
                 PT3  =  b'47d31c0a851c88f4e977f3d42a555114'
                 CT3  =  b'93d554b47b4b6561e7295ff3c95df963'
                 AT1  =  b'131527259dc7975992e4bbfd0510e9fa'
                  MT  =  b'a0fa43f94c4171977aa2a760511e3475'
                  IT  =  b'b3ef64dcd186e6cee8461c9d540edd8f'
             PADINFO  =  b'80010000800000008001000080000000'
authentication tag =  b'bf5292625deaa4a645b78d47902ef71f'
out                   =  b'af06863bfe5ab6f4d07ef32afba1baeaecd2adc6b87c84f9a
    ↪ 9f079b100f5bc9638d4e578462b696ca7aed596e3fd14e3bf5292625deaa4a645b7
    ↪ 8d47902ef71f'
```

```
m=3,a=2
CSPAE - encryption
key                   =  b'00000000000000000000000000000001'
nonce                 =  b'00000000000000000000000000000002'
message               =  b'000000000000000000000000000000030000000000000000
    ↪ 000000000000000400000000000000000000000000000005'
associated data       =  b'000000000000000000000000000000060000000000000000
```

```
    ↪ 000000000000007'
               PT0 =  b'd0017f493d6d576c9ea7d5683209ff1b'
               CT0 =  b'd0017f493d6d576c9ea7d5683209ff18'
               P0  =  b'00000000000000000000000000000003'
               C0  =  b'af06863bfe5ab6f4d07ef32afba1baea'
               PT1 =  b'7f07f972c337e1984ed92642c9a845f1'
               CT1 =  b'00000000000000000000000000000003'
               P1  =  b'00000000000000000000000000000004'
               C1  =  b'ecd2adc6b87c84f9a9f079b100f5bc96'
               PT2 =  b'ecd2adc6b87c84f9a9f079b100f5bc91'
               CT2 =  b'7f07f972c337e1984ed92642c9a845f2'
               P2  =  b'00000000000000000000000000000005'
               C2  =  b'38d4e578462b696ca7aed596e3fd14e3'
               PT3 =  b'47d31c0a851c88f4e977f3d42a555114'
               CT3 =  b'93d554b47b4b6561e7295ff3c95df963'
               AT1 =  b'131527259dc7975992e4bbfd0510e9fa'
               AT2 =  b'dba2c410fd3d0a66d02984fc6e85d61a'
                MT =  b'a0fa43f94c4171977aa2a760511e3475'
                IT =  b'7b5887e9b17c7bf1aa8b239c3f9be26f'
           PADINFO =  b'80010000000100008001000000010000'
authentication tag =  b'8896a7c8d4d6e585753fbecf68d12e69'
out                =  b'af06863bfe5ab6f4d07ef32afba1baeaecd2adc6b87c84f9a
    ↪ 9f079b100f5bc9638d4e578462b696ca7aed596e3fd14e38896a7c8d4d6e585753
    ↪ fbecf68d12e69'
```

```
m=3,a=3
CSPAE - encryption
key                =  b'00000000000000000000000000000001'
nonce              =  b'00000000000000000000000000000002'
message            =  b'0000000000000000000000000000000300000000000000000
    ↪ 00000000000000400000000000000000000000000000005'
associated data    =  b'0000000000000000000000000000000600000000000000000
    ↪ 00000000000000700000000000000000000000000000008'
               PT0 =  b'd0017f493d6d576c9ea7d5683209ff1b'
               CT0 =  b'd0017f493d6d576c9ea7d5683209ff18'
               P0  =  b'00000000000000000000000000000003'
               C0  =  b'af06863bfe5ab6f4d07ef32afba1baea'
               PT1 =  b'7f07f972c337e1984ed92642c9a845f1'
               CT1 =  b'00000000000000000000000000000003'
               P1  =  b'00000000000000000000000000000004'
               C1  =  b'ecd2adc6b87c84f9a9f079b100f5bc96'
               PT2 =  b'ecd2adc6b87c84f9a9f079b100f5bc91'
               CT2 =  b'7f07f972c337e1984ed92642c9a845f2'
               P2  =  b'00000000000000000000000000000005'
               C2  =  b'38d4e578462b696ca7aed596e3fd14e3'
               PT3 =  b'47d31c0a851c88f4e977f3d42a555114'
               CT3 =  b'93d554b47b4b6561e7295ff3c95df963'
               AT1 =  b'131527259dc7975992e4bbfd0510e9fa'
               AT2 =  b'dba2c410fd3d0a66d02984fc6e85d61a'
               AT3 =  b'7152d55728d8c03e547ab64cd21531fa'
                MT =  b'a0fa43f94c4171977aa2a760511e3475'
                IT =  b'd1a896ae6499b1a92ed8112c830b058f'
```

```
           PADINFO =  b'80010000800100008001000080010000'
authentication tag =  b'1b2c40d4b921b5fea3a2c773367276b3'
out                =  b'af06863bfe5ab6f4d07ef32afba1baeaecd2adc6b87c84f9a
    ↪ 9f079b100f5bc9638d4e578462b696ca7aed596e3fd14e31b2c40d4b921b5fea3a2
    ↪ c773367276b3'
CSPAE - decryption
key                =  b'00000000000000000000000000000001'
nonce              =  b'00000000000000000000000000000002'
message            =  b'af06863bfe5ab6f4d07ef32afba1baeaecd2adc6b87c84f9a
    ↪ 9f079b100f5bc9638d4e578462b696ca7aed596e3fd14e3'
associated data    =  b'000000000000000000000000000000060000000000000000
    ↪ 000000000000000700000000000000000000000000000008'
               PT0 =  b'd0017f493d6d576c9ea7d5683209ff1b'
               CT0 =  b'd0017f493d6d576c9ea7d5683209ff18'
               P0  =  b'00000000000000000000000000000003'
               C0  =  b'af06863bfe5ab6f4d07ef32afba1baea'
               PT1 =  b'7f07f972c337e1984ed92642c9a845f1'
               CT1 =  b'00000000000000000000000000000003'
               P1  =  b'00000000000000000000000000000004'
               C1  =  b'ecd2adc6b87c84f9a9f079b100f5bc96'
               PT2 =  b'ecd2adc6b87c84f9a9f079b100f5bc91'
               CT2 =  b'7f07f972c337e1984ed92642c9a845f2'
               P2  =  b'00000000000000000000000000000005'
               C2  =  b'38d4e578462b696ca7aed596e3fd14e3'
               PT3 =  b'47d31c0a851c88f4e977f3d42a555114'
               CT3 =  b'93d554b47b4b6561e7295ff3c95df963'
               AT1 =  b'131527259dc7975992e4bbfd0510e9fa'
               AT2 =  b'dba2c410fd3d0a66d02984fc6e85d61a'
               AT3 =  b'7152d55728d8c03e547ab64cd21531fa'
                MT =  b'a0fa43f94c4171977aa2a760511e3475'
                IT =  b'd1a896ae6499b1a92ed8112c830b058f'
           PADINFO =  b'80010000800100008001000080010000'
authentication tag =  b'1b2c40d4b921b5fea3a2c773367276b3'
provided tag       =  b'1b2c40d4b921b5fea3a2c773367276b3'
out                =  b'000000000000000000000000000000030000000000000000
    ↪ 000000000000000400000000000000000000000000000005'
```

```
m=3,a=3 padded
CSPAE - encryption
key                =  b'00000000000000000000000000000001'
nonce              =  b'00000000000000000000000000000002'
message            =  b'000000000000000000000000000000030000000000000000
    ↪ 000000000000000409'
associated data    =  b'000000000000000000000000000000060000000000000000
    ↪ 000000000000000070a0b'
               PT0 =  b'd0017f493d6d576c9ea7d5683209ff1b'
               CT0 =  b'd0017f493d6d576c9ea7d5683209ff18'
               P0  =  b'00000000000000000000000000000003'
               C0  =  b'af06863bfe5ab6f4d07ef32afba1baea'
               PT1 =  b'7f07f972c337e1984ed92642c9a845f1'
               CT1 =  b'00000000000000000000000000000003'
               P1  =  b'00000000000000000000000000000004'
```

```
                C1   =  b'ecd2adc6b87c84f9a9f079b100f5bc96'
                PT2  =  b'ecd2adc6b87c84f9a9f079b100f5bc91'
                CT2  =  b'7f07f972c337e1984ed92642c9a845f2'
                P2   =  b'09000000000000000000000000000000'
                C2   =  b'a5405b16f5db2622e1c90deba9f25963'
                PT3  =  b'd347a26436ecc7baaf102ba9605a1c91'
                CT3  =  b'93d554b47b4b6561e7295ff3c95df963'
                AT1  =  b'131527259dc7975992e4bbfd0510e9fa'
                AT2  =  b'dba2c410fd3d0a66d02984fc6e85d61a'
                AT3  =  b'd23963e8cabeaa4a76899b5f5e3ee58d'
                 MT  =  b'346efd97ffb13ed93cc57f1d1b1179f0'
                 IT  =  b'e6579e7f350f94934a4ce442452f9c7d'
            PADINFO  =  b'08010000100100000801000010010000'
authentication tag   =  b'e6b45ced002704ca27ac396b78007bd9'
out                  =  b'af06863bfe5ab6f4d07ef32afba1baeaecd2adc6b87c84f9a
    ↪ 9f079b100f5bc96a5405b16f5db2622e1c90deba9f25963e6b45ced002704ca27ac
    ↪ 396b78007bd9'
CSPAE - decryption
key                  =  b'00000000000000000000000000000001'
nonce                =  b'00000000000000000000000000000002'
message              =  b'af06863bfe5ab6f4d07ef32afba1baeaecd2adc6b87c84f9a
    ↪ 9f079b100f5bc96a5405b16f5db2622e1c90deba9f25963'
associated data      =  b'000000000000000000000000000000060000000000000000
    ↪ 0000000000000070a0b'
                PT0  =  b'd0017f493d6d576c9ea7d5683209ff1b'
                CT0  =  b'd0017f493d6d576c9ea7d5683209ff18'
                P0   =  b'00000000000000000000000000000003'
                C0   =  b'af06863bfe5ab6f4d07ef32afba1baea'
                PT1  =  b'7f07f972c337e1984ed92642c9a845f1'
                CT1  =  b'00000000000000000000000000000003'
                P1   =  b'00000000000000000000000000000004'
                C1   =  b'ecd2adc6b87c84f9a9f079b100f5bc96'
                PT2  =  b'ecd2adc6b87c84f9a9f079b100f5bc91'
                CT2  =  b'7f07f972c337e1984ed92642c9a845f2'
                P2   =  b'09000000000000000000000000000000'
                C2   =  b'a5405b16f5db2622e1c90deba9f25963'
                PT3  =  b'd347a26436ecc7baaf102ba9605a1c91'
                CT3  =  b'93d554b47b4b6561e7295ff3c95df963'
                AT1  =  b'131527259dc7975992e4bbfd0510e9fa'
                AT2  =  b'dba2c410fd3d0a66d02984fc6e85d61a'
                AT3  =  b'd23963e8cabeaa4a76899b5f5e3ee58d'
                 MT  =  b'346efd97ffb13ed93cc57f1d1b1179f0'
                 IT  =  b'e6579e7f350f94934a4ce442452f9c7d'
            PADINFO  =  b'08010000100100000801000010010000'
authentication tag   =  b'e6b45ced002704ca27ac396b78007bd9'
provided tag         =  b'e6b45ced002704ca27ac396b78007bd9'
out                  =  b'000000000000000000000000000000030000000000000000
    ↪ 00000000000000409'
```

```
CSPAE - encryption
key                  =  b'000102030405060708090a0b0c0d0e0f'
nonce                =  b'000102030405060708090a0b0c0d0e0f'
```

```
message            =
associated data    =
               PT0 =  b'c6a13b37878f5b826f4f8162a1c8d879'
               CT0 =  b'c6a13b37878f5b826f4f8162a1c8d879'
                MT =  b'fffefdfcfbfaf9f8f7f6f5f4f3f2f1f0'
                IT =  b'fffefdfcfbfaf9f8f7f6f5f4f3f2f1f0'
           PADINFO =  b'00000000000000000000000000000000'
authentication tag =  b'7525d79334164e254cba038b814d9c20'
out                =  b'7525d79334164e254cba038b814d9c20'
```

```
CSPAE - encryption
key                =  b'000102030405060708090a0b0c0d0e0f'
nonce              =  b'000102030405060708090a0b0c0d0e0f'
message            =  b'000102030405060708090a0b0c0d0e0f'
associated data    =  b'000102030405060708090a0b0c0d0e0f'
               PT0 =  b'c6a13b37878f5b826f4f8162a1c8d879'
               CT0 =  b'c6a13b37878f5b826f4f8162a1c8d879'
               P0  =  b'000102030405060708090a0b0c0d0e0f'
               C0  =  b'732b2b535f23f219b6ffc139248d2dc2'
               PT1 =  b'b58b1267dca9af9cd1b94a508948fbb4'
               CT1 =  b'00000000000000000000000000000000'
               AT1 =  b'0a940bb5416ef045f1c39458c653ea5a'
                MT =  b'b58b1267dca9af9cd1b94a508948fbb4'
                IT =  b'bf1f19d29dc75fd9207ade084f1b11ee'
           PADINFO =  b'80000000800000008000000080000000'
authentication tag =  b'0a1315ef625aedc8e354116928defef3'
out                =  b'732b2b535f23f219b6ffc139248d2dc20a1315ef625aedc8e
    ↪ 354116928defef3'
```

```
CSPAE - encryption
key                =  b'000102030405060708090a0b0c0d0e0f'
nonce              =  b'000102030405060708090a0b0c0d0e0f'
message            =  b'000102030405060708090a0b0c0d0e0f10111213141516171
    ↪ 8191a1b1c1d1e1f'
associated data    =  b'000102030405060708090a0b0c0d0e0f10111213141516171
    ↪ 8191a1b1c1d1e'
               PT0 =  b'c6a13b37878f5b826f4f8162a1c8d879'
               CT0 =  b'c6a13b37878f5b826f4f8162a1c8d879'
               P0  =  b'000102030405060708090a0b0c0d0e0f'
               C0  =  b'732b2b535f23f219b6ffc139248d2dc2'
               PT1 =  b'b58b1267dca9af9cd1b94a508948fbb4'
               CT1 =  b'00000000000000000000000000000000'
               P1  =  b'101112131415161718191a1b1c1d1e1f'
               C1  =  b'b85c0d5fa953bf572a125c9479b2e862'
               PT2 =  b'a84d1f4cbd46a940320b468f65aff67d'
               CT2 =  b'b58b1267dca9af9cd1b94a508948fbb4'
               AT1 =  b'0a940bb5416ef045f1c39458c653ea5a'
               AT2 =  b'68e2c19fa9096698ae35d29b54b0c601'
                MT =  b'79f4551c340e52f4878054e8b90659e1'
                IT =  b'111694839d07346c29b58673edb69fe0'
           PADINFO =  b'00010000f800000000010000f8000000'
authentication tag =  b'6918ce72c046c8e5159254cfe2065600'
```

```
out               =  b'732b2b535f23f219b6ffc139248d2dc2b85c0d5fa953bf572
    ↪ a125c9479b2e8626918ce72c046c8e5159254cfe2065600'
```

```
CSPAE - encryption
key               =  b'000102030405060708090a0b0c0d0e0f'
nonce             =  b'000102030405060708090a0b0c0d0e0f'
message           =  b'000102030405060708090a0b0c0d0e0f10111213141516171
    ↪ 8191a1b1c1d1e1f'
associated data   =  b'000102030405060708090a0b0c0d0e0f10111213141516171
    ↪ 8191a1b1c1d1e1f'
               PT0 =  b'c6a13b37878f5b826f4f8162a1c8d879'
               CT0 =  b'c6a13b37878f5b826f4f8162a1c8d879'
               P0  =  b'000102030405060708090a0b0c0d0e0f'
               C0  =  b'732b2b535f23f219b6ffc139248d2dc2'
               PT1 =  b'b58b1267dca9af9cd1b94a508948fbb4'
               CT1 =  b'00000000000000000000000000000000'
               P1  =  b'101112131415161718191a1b1c1d1e1f'
               C1  =  b'b85c0d5fa953bf572a125c9479b2e862'
               PT2 =  b'a84d1f4cbd46a940320b468f65aff67d'
               CT2 =  b'b58b1267dca9af9cd1b94a508948fbb4'
               AT1 =  b'0a940bb5416ef045f1c39458c653ea5a'
               AT2 =  b'3cf456b4ca488aa383c79c98b34797cb'
                MT =  b'79f4551c340e52f4878054e8b90659e1'
                IT =  b'450003a8fe46d8570447c8700a41ce2a'
           PADINFO =  b'00010000000010000000100000000010000'
authentication tag =  b'5135faad34fa275762e1dc2399a40705'
out               =  b'732b2b535f23f219b6ffc139248d2dc2b85c0d5fa953bf572
    ↪ a125c9479b2e8625135faad34fa275762e1dc2399a40705'
```

## 7.3 Rational behind the encoding of PADINFO

As explained in 4.2, the redundant encoding of `PADINFO` has been made to prevent the use of it to cancel out an error pattern introduced by a fault injection before it reaches the final $E_{\mathtt{KN}}$ operation. We made the choice of the encoding assuming that $E$ is $AES$. It does not mean that this encoding is inneficient for other block ciphers, it just means that it is adequate for $AES$.

### 7.3.1 Faults on the penultimate round of $AES$

A fault in the sbox 0 in the penultimate round of $AES$ could disturb byte 0 and 5 which are easy to cancel out via `PADINFO`, but the fault would also disturb bytes 10 and 15. Setting a non zero value on byte 15 using `PADINFO` means feeding between $2^{56}$ and $2^{64}$ bits to the algorithm. Clearly such approach is impractical on embedded systems.

### 7.3.2 Faults on the last round of $AES$

A fault on the last round of AES could disturb a single byte. The encoding of `PADINFO` make also such single byte modification difficult: changing the length of the message changes at least two bytes, the same goes for associated data. Changing both the message and the associated data length can achieve a single byte change however it requires feeding $2^{32}$ bits. This is a costly proposition for a guess trial of a fault pattern which provide only limited information about the key. High value targets can choose to limit both the $ml$ and $adl$ to $2^{32}$, in this case it is simply not possible to achieve single byte change in `PADINFO`.

## 7.4 Rational behind the initial values

The security proofs of the scheme rely on the secrecy of $\mathtt{PT}_0$ and $\mathtt{CT}_0$. Clearly $\mathtt{PT}_0$ and $\mathtt{CT}_0$ shall be different since $\mathtt{CT}_1 = \mathtt{PT}_0 \oplus \mathtt{CT}_0$ and that the security proofs of the scheme also rely on $\mathtt{CT}_1$ being secret (see for example proposition 1). To follow Kerckhoffs's principle [27], the only option is to derive them from the secret key input.

One easy solution would be to simply consider them as additional key bits, so for SPAE-AES-128 we would have a 3*128 bits key (the implementation shall then reject to execute when $\mathtt{PT}_0 == \mathtt{CT}_0$). System architect may well make this choice in the end however we felt it is desirable to limit the key material to the block cipher key so that SPAE can be used as a drop in replacement of some other AEAD in existing applications.

Another easy solution is to encrypt a constant, the firsts digit of $\pi$ for example, two times:

$$\begin{aligned} \mathtt{CST} &= 0\mathrm{x}243f6a8885a308d313198a2e03707344 \\ \mathtt{CT}_0 &= E_{\mathtt{K}}(\mathtt{CST}) \\ \mathtt{PT}_0 &= E_{\mathtt{K}}(E_{\mathtt{K}}(\mathtt{CST})) \end{aligned} \tag{1}$$

Since the calls to $E$ do not depends on $\mathtt{NONCE}$, they can therefore be precomputed. With $\mathtt{NONCE} = 0$ and $a = 0$, this would give the following equations for $\mathtt{C}_0$, $\mathtt{PT}_1$, $\mathtt{CT}_1$ and $\mathtt{TAG}_{null}$:

$$\begin{aligned} \mathtt{C}_0 &= E_{\mathtt{KN}}(E_{\mathtt{K}}(E_{\mathtt{K}}(\mathtt{CST})) \oplus \mathtt{P}_0) \oplus E_{\mathtt{K}}(\mathtt{CST}) \\ \mathtt{CT}_1 &= E_{\mathtt{K}}(E_{\mathtt{K}}(\mathtt{CST})) \oplus E_{\mathtt{K}}(\mathtt{CST}) \\ \mathtt{PT}_1 &= E_{\mathtt{KN}}(E_{\mathtt{K}}(E_{\mathtt{K}}(\mathtt{CST})) \oplus \mathtt{P}_0) \oplus \mathtt{P}_0 \\ \mathtt{TAG}_{null} &= E_{\mathtt{K}}(E_{\mathtt{K}}(\mathtt{CST})) \oplus E_{\mathtt{K}}(\mathtt{K} \oplus \mathtt{FF}) \end{aligned} \tag{2}$$

Nothing cancels out completly so this is a fine solution.

Even though the calls to $E$ can be precomputed, in practice the application may not do so. For example if the only permanent on-chip storage is a small OTP memory in which the bits are counted, $\mathtt{PT}_0$ and $\mathtt{CT}_0$ will be computed at least at each boot or even at each AEAD call. This makes this solution not ideally suited for low cost applications. We therefore searched for a solution with a single call to $E$.

As a reminder, the chosen equations for $\mathtt{PT}_0$ and $\mathtt{CT}_0$ are the following:

$$\begin{aligned} \mathtt{CT}_0 &= E_{\mathtt{K}}(\mathtt{K}) \\ \mathtt{PT}_0 &= E_{\mathtt{K}}(\mathtt{K}) \oplus \mathtt{K} \end{aligned} \tag{3}$$

Since the call to $E$ does not depends on $\mathtt{NONCE}$, it can therefore be precomputed. With $\mathtt{NONCE} = 0$ and $a = 0$, this gives the following equations for $\mathtt{C}_0$, $\mathtt{PT}_1$, $\mathtt{CT}_1$ and $\mathtt{TAG}_{null}$:

$$\begin{aligned} \mathtt{C}_0 &= E_{\mathtt{KN}}(E_{\mathtt{K}}(\mathtt{K}) \oplus \mathtt{K} \oplus \mathtt{P}_0) \oplus E_{\mathtt{K}}(\mathtt{K}) \\ \mathtt{CT}_1 &= E_{\mathtt{K}}(\mathtt{K}) \oplus \mathtt{K} \oplus E_{\mathtt{K}}(\mathtt{K}) \\ &= \mathtt{K} \\ \mathtt{PT}_1 &= E_{\mathtt{KN}}(E_{\mathtt{K}}(\mathtt{K}) \oplus \mathtt{K} \oplus \mathtt{P}_0) \oplus \mathtt{P}_0 \\ \mathtt{TAG}_{null} &= \mathtt{PT}_0 \oplus E_{\mathtt{K}}(\mathtt{AT}_0 \oplus \mathtt{K} \oplus \mathtt{FF} \oplus \mathtt{PADINFO}) \\ &= \mathtt{K} \oplus E_{\mathtt{K}}(\mathtt{K}) \oplus E_{\mathtt{K}}(\mathtt{K} \oplus \mathtt{FF}) \end{aligned} \tag{4}$$

Nothing cancels out completly so this is a fine solution. The fact that $\mathtt{CT}_1 = \mathtt{K}$ is somewhat unfortunate but we did not find any way to exploit this so far. If there is an attack exploiting this relation, SPAE and CSPAE could still be used with the more expensive alternatives for initial values that we described in this section.

*Remark* 10. It can be pointed out that in the computation of $E_{\mathtt{K}}(\mathtt{K})$ with $E = AES_{128}$ we always have at the end of the first round of $AES$ the null constant. In this case this is not a problem because we use $AES$ here merely to derive a secret from another one. The full security level of $AES$ is not required for this operation, we could have used $GCM$'s $GHASH$ or even a $CRC$ instead of $AES$. For the performance of the scheme we highlight the fact that we choose to present a scheme which uses an optimised primitive and thus using another function to derive those constants would have impact on the performances or the cost of the implementation. The storage of constants (such as $\pi$) which could be a solution to this fact has also drawbacks in the context of memory constrained embedded systems.

Nonce Misuse Resilience, Execute in Place (XIP), Differential Fault Analysis (DFA), AES, low-cost hardware,,