COMET: COunter Mode Encryption with authentication Tag

Designers/Submitters:

Shay Gueron[1,2], Ashwin Jha[3], Mridul Nandi[3]
{shay.gueron,ashwin.jha1991,mridul.nandi}@gmail.com

[1]University of Haifa, Israel
[2]Amazon Web Services Inc., Seattle, USA
[3]Indian Statistical Institute Kolkata, India

February 25, 2019

1 Specification

COunter Mode Encryption with authentication Tag, or COMET in abbreviation, is a block cipher mode of operation that provides authenticated encryption with associated data (henceforth "AEAD") functionality. At a very high level, it can be viewed as a mixture of CTR [1] and Beetle [2, 3] modes of operation. In this section we provide complete specification of the COMET family of AEAD ciphers. We first explain the COMET mode of operation, and then describe our concrete submissions based on this mode.

1.1 Notations and Conventions

We fix positive even integers n, r, κ, and t to denote the *block size*, *nonce size*, *key size*, and *tag size*, respectively in bits. We fix $p = \kappa/2$. E-n/κ denotes a block cipher family E, parametrized by the block length n and key length κ. We use $\{0,1\}^+$ and $\{0,1\}^n$ to denote the set of all non-empty (binary) strings, and n-bit strings, respectively. $\perp$ denotes the empty string and $\{0,1\}^* = \{0,1\}^+ \cup \{\perp\}$. For all practical purposes: we use little-endian format of indexing, and assume all binary strings are *byte-oriented*, i.e. belong in $(\{0,1\}^8)^*$. For any string $B \in \{0,1\}^+$, $|B|$ denotes the number of bits in B, and for $0 \leq i \leq |B|-1$, b_i denotes the i-th bit of B, i.e. $B = b_{|B|-1} \cdots b_0$. For $B \in \{0,1\}^+$, $(B_{\ell-1}, \ldots, B_0) \xleftarrow{n} B$, denotes the n-bit *block parsing* of B into $(B_{\ell-1}, \ldots, B_0)$, where $|B_i| = n$ for $0 \leq i \leq \ell-2$, and $1 \leq |B_{\ell-1}| \leq n$. For $A, B \in \{0,1\}^+$, and $|A| = |B|$, $A \oplus B$ denotes the "bitwise XOR" operation on A and B. For $A, B \in \{0,1\}^+$, $A \| B$ denotes the "string concatenation" operation on A and B. For any $B \in \{0,1\}^+$ and a non-negative integer s, $B \ll s$ and $B \lll s$ denote the "left shift by s" and "circular left shift by s" operations on B, respectively. The notations for right shift and circular right shift are analogously defined using $\gg$ and $\ggg$, respectively.

The set $\{0,1\}^p$ can be viewed as the finite field $\mathbb{F}_{2^p}$ consisting of 2^p elements. We interchangeably think of an element $B \in \mathbb{F}_{2^p}$ in any of the following ways: (i) as a p-bit string $b_{p-1} \ldots b_1 b_0 \in \{0,1\}^p$; (ii) as a polynomial $B(x) = b_{p-1}x^{p-1} + b_{p-2}x^{p-2} + \cdots + b_1 x + b_0$ over the field $\mathbb{F}_2$; (iii) a non-negative integer $b < 2^p$; (iv) an abstract element in the field. Addition in $\mathbb{F}_{2^p}$ is just bitwise XOR of two p-bit strings, and hence denoted by $\oplus$. $P(x)$ denotes the primitive polynomial used to represent the field $\mathbb{F}_{2^p}$, and α denotes a fixed primitive element in this representation. The multiplication of $A, B \in \mathbb{F}_{2^p}$ is defined as $A \odot B := A(x) \cdot B(x) \pmod{P(x)}$, i.e. polynomial multiplication modulo $P(x)$ in $\mathbb{F}_2$. For any $B \in \mathbb{F}_{2^p}$, multiplication with α is computationally efficient. We demonstrate this for $p = 64$, as we will fix $\kappa = 128$ in our submissions. For $p = 64$, $P(x) = x^{64} + x^4 + x^3 + x + 1$ is a primitive polynomial, and we let α to denote the primitive element $2 \in \mathbb{F}_{2^{64}}$. Then for any $B \in \mathbb{F}_{2^{64}}$, we have

$$A \odot \alpha = \begin{cases} A \ll 1 & \text{if } a_{|A|-1} = 0, \\ (A \ll 1) \oplus 0^{59}11011 & \text{if } a_{|A|-1} = 1. \end{cases}$$

1.2 Parameters

COMET is primarily parameterized by the block size n of the underlying block cipher, where $n \in \{64, 128\}$. In other words we allow block ciphers with 64-bit and 128-bit block sizes. We simply write COMET-n to denote COMET with the particular choice of n. The secondary parameters are set according to the value of n in the following manner.

- COMET-128: In this version $n = 128$, $r = 128$, $\kappa = 128$, $t = 128$, and $p = 64$.
- COMET-64: In this version $n = 64$, $r = 120$, $\kappa = 128$, $t = 64$, and $p = 64$.

In both variants, we use the primitive polynomial $P(x) = x^{64} + x^4 + x^3 + x + 1$ to represent the field $\mathbb{F}_{2^{64}} = \{0,1\}^{64}$, and fix the primitive element $\alpha = 2$.

1.3 Description of COMET

Algorithms 1-3 give the complete algorithmic description of the mode, and figure 1 illustrates the major components of the encryption/decryption process. In the remainder of this subsection, we give a high level description of the main modules (given in algorithm 2) used in the encryption/decryption (described in algorithm 1) process.

- init: Apart from some book-keeping operations, the major task of this module is to create the initial state using the public nonce N and the secret key K. This initial state derivation is the only stage where the two versions of COMET, namely, COMET-128 and COMET-64 vary. The state can be viewed as an $(n+\kappa)$-bit concatenated string $Y \| Z$ made up of n-bit string Y (also called the Y-state) and κ-bit string Z (also called Z-state). In this notation the initial state is $(Y_0, Z_0) = Y_0 \| Z_0$. In case of COMET-128,

we define $Y_0 = K$ and $Z_0 = E_K(N)$, as described in "**function** init_state_128" of algorithm 3. In case of COMET-64, we use $Y_0 = E_K(0)$ and $Z_0 = K \oplus 0^8 \| N$, as described in "**function** init_state_64" of algorithm 3.

- proc_ad: This module is responsible for the associated data (AD) processing. At the start of the processing a control bit indicating start of non-empty AD is XORed to the 5^{th} most significant bit (msb) of the current Z-state. The AD data is absorbed, n bits at a time, using "**function** round" of algorithm 2. In case of partial last block a control bit indicating partial block is XORed to the 4^{th} msb of the Z-state before the processing of the last block.
- proc_pt: This module is responsible for the plaintext (PT) processing. At the start of the processing a control bit indicating start of non-empty PT is XORed to the 3^{rd} msb of the current Z-state. PT processing is similar to AD processing except for the fact that we squeeze out n-bit ciphertext as well. In case of partial last block a control bit indicating partial block is XORed to the 2^{nd} msb of the Z-state before the processing of the last block.
- proc_ct: This module is responsible for ciphertext (CT) processing. It is symmetrical to proc_pt.
- proc_tg: This module is responsible for tag generation. Before the tag generation a control bit indicating the tag generation call is XORed to the msb of the current Z-state.

Algorithm 1 Encryption/Decryption algorithm in COMET.

function COMET-$n_{[E]}$.enc(K, N, A, M)
  $C \leftarrow \perp$
  $(Y_0, Z_0, a, m, \ell) \leftarrow$ init(K, N, A, M)
  if $a \neq 0$ **then**
    $(Y_a, Z_a) \leftarrow$ proc_ad(Y_0, Z_0, A)
  if $m \neq 0$ **then**
    $(Y_\ell, Z_\ell, C) \leftarrow$ proc_pt(Y_a, Z_a, M)
  $T \leftarrow$ proc_tg(Y_ℓ, Z_ℓ)
  return (C, T)

function COMET-$n_{[E]}$.dec(K, N, A, C, T)
  $M \leftarrow \perp$
  is_auth $\leftarrow 0$
  $(Y_0, Z_0, a, m, \ell) \leftarrow$ init(K, N, A, C)
  if $a \neq 0$ **then**
    $(Y_a, Z_a) \leftarrow$ proc_ad(Y_0, Z_0, A)
  if $m \neq 0$ **then**
    $(Y_\ell, Z_\ell, M) \leftarrow$ proc_ct(Y_a, Z_a, C)
  $T' \leftarrow$ proc_tg(Y_ℓ, Z_ℓ)
  if $T' = T$ **then**
    is_auth $\leftarrow 1$
  else
    $M \leftarrow \perp$
  return (is_auth, M)

1.4 Description of Block Ciphers

Here we give a concise description of, AES-128/128, CHAM, and Speck-64/128, the three block ciphers used in our submissions. These block ciphers are already published in [4, 5, 6, 7], and freely accessible at [4, 5, 8]. We remark that we reuse exactly the same description of AES-128/128 [4], CHAM [5], and Speck-64/128 [6, 7], and the description here is just for the sake of completeness. We also note that, we only present the encryption function, as the decryption functions is not required in our AEAD algorithms. For detailed description and rationale, the readers are referred to the original specifications given in [4] for AES-128/128, [5] for CHAM-128/128 and CHAM-64/128, and [6, 7] for Speck-64/128.

1.4.1 Description of AES-128/128

AES [4] is based on the substitution-permutation network or SPN design paradigm. It has a fixed block size of 128 bits, and the key size can be 128, 192, or 256 bits. We only use the variant with 128-bit key size, i.e. AES-128/128.

The algorithm consists of 10 rounds, composed of four main steps: `SubBytes`, `ShiftRows`, `MixColumns`, and `AddRoundKey`. The 128-bit internal state of AES-128/128 is also viewed as a 4×4 matrix over $\mathbb{F}_{2^8}$ in column-major order. For example, let $S_{15}, \ldots, S_1, S_0$ be some internal state, then it is viewed as:

$$\begin{bmatrix} S_0 & S_4 & S_8 & S_{12} \\ S_1 & S_5 & S_9 & S_{13} \\ S_2 & S_6 & S_{10} & S_{14} \\ S_3 & S_7 & S_{11} & S_{15} \end{bmatrix}$$

Now, the encryption algorithm can be described by the following steps in sequence:

Algorithm 2 Main modules of COMET.

function init(K, N, A, M)
  if $n = 64$ **then**
    $(Y_0, Z_0) \leftarrow$ init_state_64(K, N)
  else
    $(Y_0, Z_0) \leftarrow$ init_state_128(K, N)
  $a \leftarrow \lceil |A|/n \rceil$
  $m \leftarrow \lceil |M|/n \rceil$
  $\ell \leftarrow a + m$
  return (Y_0, Z_0, a, m, ℓ)

function round(Y', Z', I, b)
  $Z \leftarrow$ get_blk_key(Z')
  $X \leftarrow \mathsf{E}(Z, Y')$
  if $b = 0$ **then**
    $Y \leftarrow$ update$(X, I, 0)$
    return (Y, Z)
  else
    $(Y, O) \leftarrow$ update(X, I, b)
    return (Y, Z, O)

function proc_ad(Y_0, Z_0, A)
  $(A_{a-1}, \ldots, A_0) \leftarrow$ parse(A)
  $Z_0 \leftarrow Z_0 \oplus 000010^{\kappa-5}$
  for $i = 0$ **to** $a - 2$ **do**
    $(Y_{i+1}, Z_{i+1}) \leftarrow$ round$(Y_i, Z_i, A_i, 0)$
  if $n \nmid |A_{a-1}|$ **then**
    $Z_{a-1} \leftarrow Z_{a-1} \oplus 000100^{\kappa-5}$
  $(Y_a, Z_a) \leftarrow$ round$(Y_{a-1}, Z_{a-1}, A_{a-1}, 0)$
  return (Y_a, Z_a)

function proc_pt(Y_a, Z_a, M)
  $(M_{m-1}, \ldots, M_0) \leftarrow$ parse(M)
  $Z_a \leftarrow Z_a \oplus 001000^{\kappa-5}$
  for $j = 0$ **to** $m - 2$ **do**
    $k \leftarrow a + j$
    $(Y_{k+1}, Z_{k+1}, C_j) \leftarrow$ round$(Y_k, Z_k, M_j, 1)$
  if $n \nmid |M_{m-1}|$ **then**
    $Z_{\ell-1} \leftarrow Z_{\ell-1} \oplus 010000^{\kappa-5}$
  $(Y_\ell, Z_\ell, C_{m-1}) \leftarrow$ round$(Y_{\ell-1}, Z_{\ell-1}, M_{m-1}, 1)$
  $C \leftarrow (C_{m-1}, \ldots, C_0)$
  return (Y_ℓ, Z_ℓ, C)

function proc_ct(Y_a, Z_a, C)
  $(C_{m-1}, \ldots, C_0) \leftarrow$ parse(C)
  $Z_a \leftarrow Z_a \oplus 001000^{\kappa-5}$
  for $j = 0$ **to** $m - 2$ **do**
    $k \leftarrow a + j$
    $(Y_{k+1}, Z_{k+1}, M_j) \leftarrow$ round$(Y_k, Z_k, C_j, 2)$
  if $n \nmid |C_{m-1}|$ **then**
    $Z_{\ell-1} \leftarrow Z_{\ell-1} \oplus 010000^{\kappa-5}$
  $(Y_\ell, Z_\ell, M_{m-1}) \leftarrow$ round$(Y_{\ell-1}, Z_{\ell-1}, C_{m-1}, 2)$
  $M \leftarrow (M_{m-1}, \ldots, M_0)$
  return (Y_ℓ, Z_ℓ, M)

function proc_tg(Y_ℓ, Z_ℓ)
  $Z_\ell \leftarrow Z_\ell \oplus 100000^{\kappa-5}$
  $Z_{\ell+1} \leftarrow$ get_blk_key(Z_ℓ)
  $T \leftarrow \mathsf{E}(Z_{\ell+1}, Y_\ell)$
  return T

1. `KeyExpansion`: As the first step, the 128-bit key is processed to derive 11 subkeys, one for each round and an extra subkey for initial key whitening. The round keys are obtained as follows: Let K be the key, and $(K_3, K_2, K_1, K_0) \xleftarrow{32} K$. For $S = (S_3, S_2, S_1, S_0) \in (\mathbb{F}_{2^8})^4$, let $\mathsf{SubWord}(S) := (\mathsf{SB}[S_3], \mathsf{SB}[S_2], \mathsf{SB}[S_1], \mathsf{SB}[S_0])$, where SB denotes the AES S-box given in Table 1 (see [4] for more details). Then, for $i \in \{0, \ldots, 43\}$, we have:

$$W_i := \begin{cases} K_i & \text{if } i < 4 \\ W_{i-4} \oplus (\mathsf{SubWord}(W_{i-1}) \ggg 8) \oplus \mathrm{R}_{i/4} & \text{if } i \geq 4 \text{ and } i \equiv 0 \bmod 4 \\ W_{i-4} \oplus W_{i-1} & \text{otherwise,} \end{cases}$$

   where R denotes the round constant array given in Table 2 (see [4] for more details). Note the direction of rotation is towards right due to the little endian format used here. For $i \in [10]$, we write K^i to denote the i-th round key $W_{4i+3}, W_{4i+2}, W_{4i+1}, W_{4i}$, and K^0 to denote the initial whitening key W_3, W_2, W_1, W_0.

2. Initial round key whitening: This step generates the initial state S^0 using `AddRoundKey` (described below) with the initial whitening key K^0 and the plaintext P.

3. The following sequence of steps constitute one AES intermediate round, which is repeated 9 times:

   (a) `SubBytes`: Let S denote the internal state at this moment (consists of the plaintext). Then this step can be described by the mapping, $S_i \mapsto \mathsf{SB}[S_i]$, for all $i \in \{0, \ldots, 15\}$, where SB denotes the AES S-box given in [4].

   (b) `ShiftRows`: The `ShiftRows` step applied to an internal state s, can be described by the following mapping:

$$\begin{bmatrix} S_0 & S_4 & S_8 & S_{12} \\ S_1 & S_5 & S_9 & S_{13} \\ S_2 & S_6 & S_{10} & S_{14} \\ S_3 & S_7 & S_{11} & S_{15} \end{bmatrix} \longmapsto \begin{bmatrix} S_0 & S_4 & S_8 & S_{12} \\ S_5 & S_9 & S_{13} & S_1 \\ S_{10} & S_{14} & S_2 & S_6 \\ S_{15} & S_3 & S_7 & S_{11} \end{bmatrix}$$

Algorithm 3 Various sub-modules of COMET.

function chop(I, ℓ)
  if $\ell > n$ **then**
    return $\perp$
  else
    return $i_{\ell-1} \dots i_0$

function parse(I)
  $\ell = \lceil |I|/n \rceil$
  if $\ell = 0$ **then**
    return $\perp$
  else
    $(I_{\ell-1}, \dots, I_0) \xleftarrow{n} I$
    return $(I_{\ell-1}, \dots, I_0)$

function opt_pad0*1(I)
  if $|I| = 0$ **or** $n \nmid |I|$ **then**
    $\xi = n - (|I| \bmod n)$
    $I \leftarrow 0^{\xi-1}1 \| I$
  return I

function permute(Z')
  $(Z'_1, Z'_0) \xleftarrow{p} Z'$
  $Z_0 \leftarrow Z'_0 \odot \alpha$
  $Z \leftarrow (Z'_1, Z_0)$
  return Z

function init_state_128(K, N)
  $Y \leftarrow K$
  $Z \leftarrow \mathsf{E}(K, N)$
  return (Y, Z)

function init_state_64(K, N)
  $Y \leftarrow \mathsf{E}(K, 0)$
  $Z \leftarrow K \oplus 0^{\kappa-r} \| N$
  return (Y, Z)

function shuffle(X')
  $(X'_3, X'_2, X'_1, X'_0) \xleftarrow{n/4} X'$
  $X_2 \leftarrow X'_2 \ggg 1$
  $X \leftarrow (X'_1, X'_0, X_2, X'_3)$
  return X

function get_blk_key(Z')
  $Z \leftarrow$ permute(Z')
  return Z

function update(X, I, b)
  if $b = 0$ **then**
    $Y \leftarrow X \oplus$ opt_pad0*1(I)
    return Y
  else
    $X' \leftarrow$ shuffle(X)
    $O \leftarrow$ chop($X', |I|$) $\oplus I$
    if $b = 1$ **then**
      $Y \leftarrow X \oplus$ opt_pad0*1(I)
    else if $b = 2$ **then**
      $Y \leftarrow X \oplus$ opt_pad0*1(O)
    return (Y, O)

(c) `MixColumns`: The `MixColumns` step applies an invertible linear transformation on each column of the state matrix. The linear transformation is described by the following matrix:

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

over the field $\mathbb{F}_{2^8}$ defined with respect to the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$.

(d) `AddRoundKey`: The `AddRoundKey` step for the i-th round and internal state s is defined by the mapping $S_j \mapsto S_j \oplus K_j^i$, for all $j \in \{0, \dots, 15\}$.

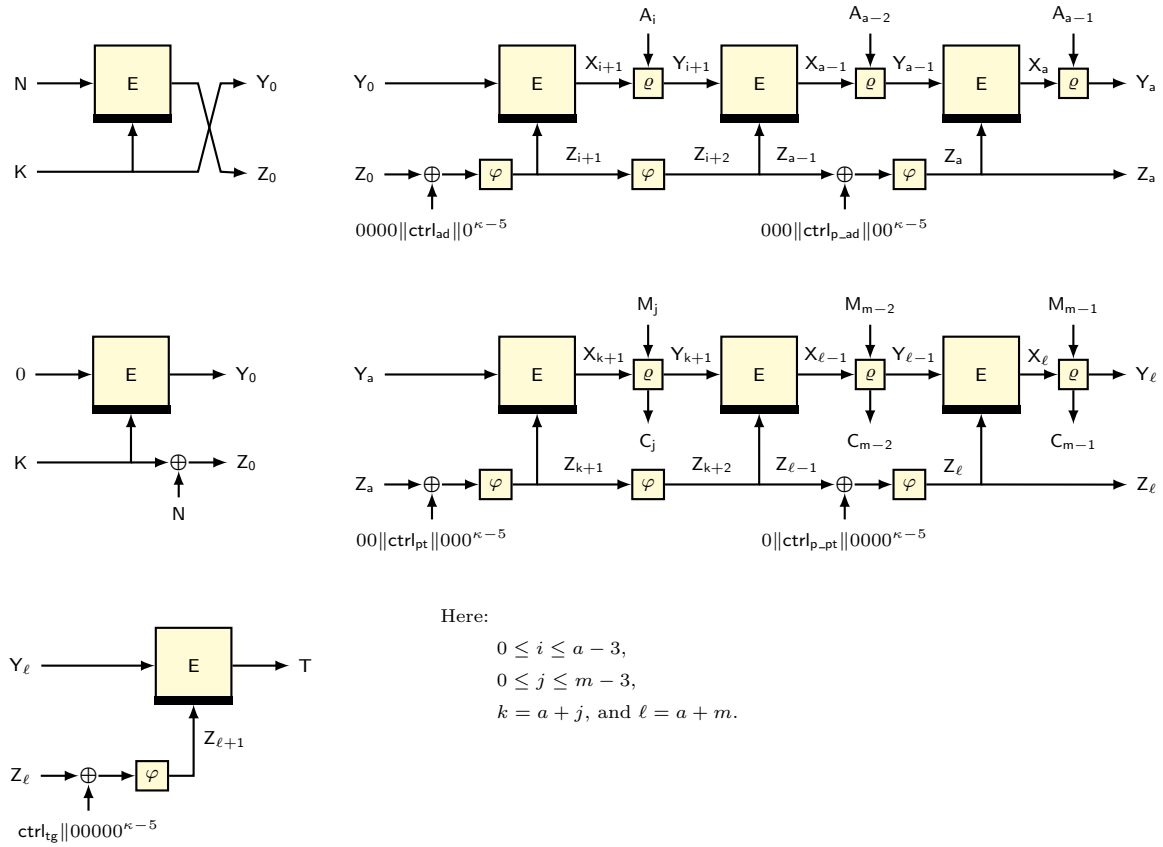
4. The final round invokes `SubBytes`, `ShiftRows`, and `AddRoundKey` in order, skipping the `MixColumns` operation.

1.4.2 Description of CHAM

CHAM [5] is based on the ARX design paradigm, and hence does not require any S-boxes. It can have block size $n \in \{64, 128\}$, and key size $\kappa \in \{n, 2n\}$. We focus on the $\kappa = n$ case. Both CHAM-128/128 and CHAM-64/128 consist of 80 rounds. CHAM-128/128 views its state as a concatenation of four 32-bit words. Similarly, CHAM-64/128 views its state as a concatenation of four 16-bit words. The number of rounds and the word size are denoted as r and w, respectively.

In the following discussion we write $a\%b$ to denote $a \mod b$, and $\boxplus$ to denote addition modulo 2^w. For $n \in \{64, 128\}$ and $\kappa = n$, CHAM-n/κ encrypts a plaintext $P \in \{0,1\}^n$ to a ciphertext $C \in \{0,1\}^n$ using a secret key $K \in \{0,1\}^\kappa$ by applying r iterations of a round function as follows:

1. First, P is parsed into four w-bit words S_3^0, S_2^0, S_1^0, S_0^0.



Here:

$0 \le i \le a-3$,
$0 \le j \le m-3$,
$k = a + j$, and $\ell = a + m$.

Figure 1: Schematic diagram of different modules used in the encryption algorithm of COMET for non empty AD and PT. From top to bottom and left to right, we have the following modules: init_state_128, init_state_64, proc_tg, proc_ad, and proc_pt. φ and ϱ denote the functional view of sub-modules permute and shuffle from algorithm 3, respectively. See algorithm 1-3 for more details.

2. For $i \in \{0, \ldots, r-1\}$, the i-th round output, S^{i+1} is computed as:

$$S_3^{i+1} = ((S_0^i \oplus i) \boxplus ((S_1^i \lll 1) \oplus R_{i\%2\kappa/w})) \lll 8$$
$$S_j^{i+1} = S_{j+1}^i \text{ for } 0 \le j \le 2,$$

if i is even, otherwise,

$$S_3^{i+1} = ((S_0^i \oplus i) \boxplus ((S_1^i \lll 8) \oplus R_{i\%2\kappa/w})) \lll 1$$
$$S_j^{i+1} = S_{j+1}^i \text{ for } 0 \le j \le 2,$$

where $RK_{i\%2k/w}$ is the round key. The ciphertext C is defined as $C = (S_3^r, S_2^r, S_1^r, S_0^r)$.

3. The key schedule of CHAM-n/κ takes the secret key $K \in \{0,1\}^\kappa$ and generates the $2\kappa/w$ w-bit round keys $R_0, R_1, \ldots, R_{2\kappa/w-1}$. Initially, divide the secret key into κ/w w-bit round words $K_0, K_1, \ldots, K_{\kappa/w-1}$. Then the round keys are generated as follows:

$$R_i = K_i \oplus (K_i \lll 1) \oplus (K_i \lll 8),$$
$$R_{(i+\kappa/w)\oplus 1} = K_i \oplus (K_i \lll 1) \oplus (K_i \lll 11),$$

where $i \in \{0, \ldots, \kappa/w\}$.

1.4.3 Description of Speck-64/128

Speck-64/128 [6, 7] is based on ARX design paradigm. Speck-64/128 allows for a number of block size/key size combinations, but we will focus on Speck-64/128[64] which requires 27 rounds of a key-dependent map $f_R : \mathbb{F}_{2^{32}} \times \mathbb{F}_{2^{32}} \to \mathbb{F}_{2^{32}} \times \mathbb{F}_{2^{32}}$ defined by

$$\forall (X_1, X_0) \in \mathbb{F}_{2^{32}} \times \mathbb{F}_{2^{32}}, \quad f_R(X_1, X_0) := (((X_1 \ggg 8) \boxplus X_0) \oplus R, ((X_0 \lll 3) \oplus ((X_1 \ggg 8) \boxplus X_0) \oplus R)),$$

Table 1: AES-128/128 S-Box Table. All values are given in hexadecimal format. The row and column denote the most and least significant nibble of the input. For example For example $3a$ is converted into 80.

	00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**0a**	**0b**	**0c**	**0d**	**0e**	**0f**
00	63	7c	77	7b	f2	6b	6f	35	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	97	26	36	3f	f7	cc	34	a5	ef	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	89	a1	89	0d	bf	e6	42	68	42	99	2d	0f	b0	54	bb	16

Table 2: AES-128/128 Round Constants Table. All values are given in hexadecimal format.

Round:	1	2	3	4	5	6	7	8	9	10
R	01	02	04	08	10	20	40	80	1b	36

where R denotes the 32-bit subkey, and (X_1, X_0) denotes the 64-bit state of the cipher. The Speck-64/128 key schedule uses similar round function to generate the round subkeys R_i. Let K be the 128-bit key of Speck-64/128[64]. We write $(K_3, K_2, K_1, K_0) \xleftarrow{32} K$. Let $L_2 = K_3$, $L_1 = K_2$, $L_0 = K_0$, and $R_0 = K_0$. Then for $0 \leq i < 27$, the round subkeys are defined as:

$$L_{i+3} = (R_i \boxplus (L_i \ggg 8)) \oplus i,$$
$$R_{i+1} = (R_i \lll 3) \oplus L_{i+3},$$

and $(R_{26}, \ldots, R_0)$ denote the round subkeys.

2 Concrete Proposals for Submission

COMET is designed to have small state and small number of operations in both hardware and software. The major chunk of computational time and power is used by the underlying block cipher. In this section we provide concrete proposals based on three block ciphers, namely, AES-128/128 [4], CHAM [5], and Speck-64/128 [6, 7, 8]. The proposals based on these block ciphers are categorized into software and hardware oriented proposals, due to their relative advantages in their respective category. Here we remark that the segregation is only for recommendation purposes, and the seven schemes listed below are suitable for application in both software and hardware.

2.1 Software Oriented Lightweight AEAD Proposals

Speck-64/128 is specially designed for applications in small micro-controllers [6] and AES-128/128 is supported on many micro-controllers and processors. So, they are well-suited for applications requiring lightweight software implementations. We propose the following submissions for applications in software oriented areas:

1. COMET-128_AES-128/128: This version sets $n = 128$ and uses the 8-round variant of AES-128/128 as the underlying block cipher. It requires a fixed length nonce of size $r = 128$. The maximum length of plaintext or associated data, and the amount of data to be processed using a single key, is bounded by at most 2^{64} bytes. The tag size $t = 128$ bits.

3. COMET-64_Speck-64/128: This version sets $n = 64$ and uses Speck-64/128 as the underlying block cipher. It requires a fixed length nonce of size $r = 120$. The maximum length of plaintext or associated

data, and the amount of data to be processed using a single key, is bounded by at most 2^{45} bytes. The tag size $t = 64$ bits.

2.2 Hardware Oriented Lightweight AEAD Proposals

CHAM is an ultra-lightweight block cipher which has very small hardware footprint [5]. We propose the following CHAM-based submissions for applications in hardware oriented areas:

1. COMET-128_CHAM-128/128: This version sets $n = 128$ and uses CHAM-128/128 as the underlying block cipher. It requires a fixed length nonce of size $r = 128$. The maximum length of plaintext or associated data, and the amount of data to be processed using a single key, is bounded by at most 2^{64} bytes. The tag size $t = 128$ bits.
2. COMET-64_CHAM-64/128: This version sets $n = 64$ and uses CHAM-64/128 as the underlying block cipher. It requires a fixed length nonce of size $r = 120$. The maximum length of plaintext or associated data, and the amount of data to be processed using a single key, is bounded by at most 2^{45} bytes. The tag size $t = 64$ bits.

2.2.1 Primary Version

We fix COMET-128_AES-128/128 as our primary submission in the AEAD category. We choose the mode COMET-128 for two reasons:

- *Better security:* It achieves better data complexity limits than COMET-64. Additionally it gives better security in random nonce scenario (see section 4).
- *Higher throughput:* The data processing rate is a healthy 128-bit per block cipher call. This could be crucial in reducing the latency and energy consumption.

We choose the block cipher AES-128/128, as it is well-analyzed in both single-key and related-key models, and hence instills greater confidence to the security of COMET-128_AES-128/128. Furthermore, AES-128/128 is a standardized block cipher, which means that most of the well-known cryptographic libraries offer an AES-128/128 implementation. This will allow for swift integration of COMET-128_AES-128/128.
Several test vectors corresponding to the four submissions are available in appendix **??**.

3 Design Rationale

The primary motivation behind COMET is the design of a lightweight (in hardware/software state/memory footprint) yet adequately efficient and secure AEAD mode of operation for block ciphers. Particularly, our goal is to keep minimal state size, and then aim for better performance and security. For this we start with the design paradigm of Beetle [2, 3], a sponge variant, and think of ways to replace the internal permutation call with a block cipher call. For $b \geq 1$, suppose, $P : \{0,1\}^b \rightarrow \{0,1\}^b$ is a permutation over $\{0,1\}^b$. Now, for $b = n + \kappa$ and $(x, z) \in \{0,1\}^n \times \{0,1\}^\kappa$, we define $P(x, z) := (\mathsf{E}(z, x), \varphi(z))$ where E is a block cipher with n-bit block size and κ-bit key size, and φ is a permutation over $\{0,1\}^\kappa$. Now if φ is light and efficient then one can expect that this definition of P will be more efficient then a larger permutation over $\{0,1\}^b$, as it may require more rounds to mix b bits. In order to keep φ light we use an encoding of the block position (motivated by the CTR mode [1]) to permute and generate the next block key. Combining the two elements: state size reduction following Beetle paradigm, and efficient and light key updation using encoding of block position following CTR paradigm, we get a mode with a small state ($(n+\kappa)$ bits) and high security (2^n data and 2^κ time complexity).

3.1 Nonce Usage

COMET makes a single pass over the input associated data and plaintext to reduce the latency in producing the ciphertext. In general, single pass AEAD's use a non-repeating nonce to generate a (possibly uniform) random state for each encryption query. COMET also follows the same paradigm and is secure in *nonce respecting* scenario, i.e. each nonce should be used to encrypt a single message in the lifetime of the key. We remark here that we do not claim any security when the nonces are reused in an arbitrary fashion to encrypt multiple messages, although birthday bound (in the block size) security is still possible for scenarios where the nonce is chosen randomly. This is mentioned in section 4.

3.2 Number of Block Cipher Calls

The cipher requires 1 call (this can be cached between queries for COMET-64, if storage is available) for initialization; a and m calls for processing AD and message, respectively, where $a = \lceil |A|/n \rceil$ and $m = \lceil |M|/n \rceil$; and 1 call for tag generation. In total, the cipher requires $a + m + 2$ many calls to process an a block AD and m block message. In the following list, we enumerate the exact number of calls for some cases.

1. For $|A| = 0$, $|M| = 0$: 2 calls — 1 call for initialization, and 1 call for tag generation.
2. For $0 < |A| \leq n$, $|M| = 0$: 3 calls — 1 call for initialization, 1 for AD processing, and 1 for tag generation.
3. For $3n < |A| \leq 4n$, $|M| = 0$: 6 calls — 1 call for initialization, 4 for AD processing, and 1 for tag generation.
4. For $|A| = 0$, $|M| \leq n$: 3 calls — 1 call for initialization, 1 for key stream generation, and 1 for tag generation.
5. For $|A| = 0$, $3n < |M| \leq 4n$: 6 calls — 1 call for initialization, 4 for key stream generation and 1 for tag generation.
6. For $0 < |A| \leq n$, $0 < |M| \leq n$: 4 calls — 1 call for initialization, 1 for AD processing, 1 for key stream generation, and 1 for tag generation.
7. For $3n < |A| \leq 4n$, $3n < |M| \leq 4n$: 10 calls — 1 call for initialization, 4 for AD processing, 4 for key stream generation, and 1 for tag generation.

3.3 Choice of init_state Function

COMET employs nonce-based initialization (see Figure 1) at the start to generate a random state (initial key and input). This is done while restricting the number of block cipher calls to 1, so as to keep minimal overhead. Depending upon the variants, namely, COMET-128 and COMET-64, we employ two different approaches in this step:

State Derivation in COMET-128: In this case the input nonce is encrypted with the master key to get a nonce derived key and the master key is used as the initial input. This helps in two respects: first, the security of the mode can be shown in more relaxed security models [9]; second, the underlying block cipher is only required to be related-key secure under without replacement key derivation.

State Derivation in COMET-64: In this case, the block size is smaller than the nonce size, which makes it difficult to process the nonce in one go. Since the focus of COMET-64 is more towards, state size reduction, we sacrifice a little bit in terms of security. Here the master key is XORed with the input nonce to generate the initial key and the encryption of 0 under master key is used as the initial input.

3.4 Choice of permute Function

We need that the permute function should have a large period, so that within an encryption query there is no collision in the key input of each block. Multiplication by primitive element α has this property. In this case two block keys can be same only if the input length goes beyond 2^{64}. So we choose α multiplication as our choice of permute. Note that, it is also expected to resist the existing related-key attack strategies based on key difference.

3.5 Choice of shuffle Function

Like Beetle [2, 3], we need that both $\mathsf{shuffle}(x)$ and $\mathsf{shuffle}(x) \oplus \mathsf{id}(x)$ should be invertible for all x, where id denotes the identity function. We choose a variant of the shuffle function used in Beetle, in order to reduce the number of shift and swap operations.

3.6 The Control Bits

Here we explain the 5-bit control signal that we use to separate the processing of various critical blocks. The control signals can be viewed as a 5-bit string described below

$$\mathsf{ctrl}_{\mathsf{tg}}\ \mathsf{ctrl}_{\mathsf{p\_pt}}\ \mathsf{ctrl}_{\mathsf{pt}}\ \mathsf{ctrl}_{\mathsf{p\_ad}}\ \mathsf{ctrl}_{\mathsf{ad}}$$

Initially, all bits are set to 0. The bits are set to 1 in the following manner:

1. $\mathsf{ctrl}_{\mathsf{ad}}$: The bit sets to 1 at first AD block processing call. For empty AD it remains set to 0.
2. $\mathsf{ctrl}_{\mathsf{p\_ad}}$: The bit sets to 1 at the last AD block processing call if the last block is partial. For full last block it remains set to 0.
3. $\mathsf{ctrl}_{\mathsf{pt}}$: The bit sets to 1 at first message block processing call. For empty messages it remains set to 0.
4. $\mathsf{ctrl}_{\mathsf{p\_pt}}$: The bit sets to 1 at the last message block processing call if the last block is partial. For full last block it remains set to 0.
5. $\mathsf{ctrl}_{\mathsf{tg}}$: The bit sets to 1 at tag generation call.

At each of the 5 critical stages, we XOR the corresponding signal to the appropriate bit (as described above) of the Z-state. Note that, in hardware implementations we do not actually need 5 signals. Indeed, one can reuse the $\mathsf{ctrl}_{\mathsf{ad}}$ and $\mathsf{ctrl}_{\mathsf{p\_ad}}$ for $\mathsf{ctrl}_{\mathsf{pt}}$ and $\mathsf{ctrl}_{\mathsf{p\_pt}}$, respectively, which means that we only need 3 signals.

4 Security Claims

Table 3: Summary of security claims for various submissions based on COMET.

Submissions	Confidentiality		Integrity	
	Time	Data (in bytes)	Time	Data (in bytes)
COMET-128_AES-128/128	2^{119}	2^{64}	2^{119}	2^{64}
COMET-128_CHAM-128/128	2^{119}	2^{64}	2^{119}	2^{64}
COMET-64_Speck-64/128	2^{119}	2^{64}	2^{112}	2^{45}
COMET-64_CHAM-64/128	2^{119}	2^{64}	2^{112}	2^{45}

In Table 3, we give a quantitative summary of the expected security level for the four instantiations of COMET. We assume a nonce-respecting environment, i.e., for a fixed key, for two distinct encryption queries, the public nonce value is always distinct, although COMET-128 could also be secure in scenarios, where the nonce is chosen uniformly from the nonce space. The security claims are based on full round AES-128/128, CHAM-128/128, CHAM-64/128, and Speck-64/128, and we do not claim any security for COMET with round-reduced variants of these block ciphers. Note that COMET-64 allows for higher data limits in confidentiality, as compared to the data limit in integrity/authenticity. We remark that we could not find any matching forging attack(s) on COMET-64, even when the integrity data limits was significantly higher than the one given in Table 3, so it is quite possible that the integrity security margins for COMET-64 could be improved.

5 Security Analysis

In this section, we give a brief and informal justification for the security claims made in Table 3 of section 4.

5.1 Security of COMET

We analyze the COMET mode of operation against generic attacks (assuming the underlying block cipher is ideal, i.e. random permutation). First we briefly explain possible attack strategies along with a rough lower bound estimate on the data and time complexity of each strategy. Then we validate the recommended criteria given in Table 3 by substituting concrete parameters. In the following discussion:

- D denotes the total (both encryption and decryption) data complexity. This parameter quantifies the online resource requirements, and includes the total number of blocks (among all messages and associated data) processed through the underlying block cipher for a fixed master key. We use D_e and D_v to account for the data complexity of encryption and decryption/verification queries.
- T denotes the time complexity. This parameter quantifies the offline resource requirements, and includes the total time required to process the offline evaluations of the underlying block cipher. Since one call of the block cipher can be assumed to take a constant amount of time, we generally take T as the total number of offline calls to the block cipher.

5.1.1 Master Key or Internal State Recovery

Master Key Recovery: The adversary can try to guess the master key using offline block cipher queries. Once the master key is known the adversary can certainly distinguish, forge valid ciphertexts, or recover the plaintext. But, since the master key is chosen uniformly in both COMET-128 and COMET-64, this strategy would require $T \approx 2^{\kappa}$ many offline queries and a constant number of encryption queries, i.e. $D_e = O(1)$.

State Recovery: The adversary can try to guess the internal state for some encrypted block using a combination of offline and online queries. If the adversary guesses the state correctly then it can forge valid ciphertexts for the nonce value used in this encrypted block. We argue that guessing the internal state is much harder than guessing the master key itself. Indeed, one can recover the key once the state is recovered. Note that guessing just one of Y or Z is not enough as the other value is random. Further, guessing both Y and Z requires the product of data and time, $D_eT \approx 2^{n+\kappa}$. This can be argued using list matching arguments, i.e. the adversary creates a list $\mathcal{L}_T$ of T offline query-response tuples and a list $\mathcal{L}_{D_e}$ of D_e online query-response tuples (with empty PT and AD). It can then try to get a matching between $\mathcal{L}_T$ and $\mathcal{L}_{D_e}$, and for each matching try an appropriate forging attempt. A matching between any element of $\mathcal{L}_T$ and any element of $\mathcal{L}_{D_e}$ would happen with approx. $2^{-(n+\kappa)}$ probability (as the key and mask are random and almost independent of each other). So we need $D_eT \approx 2^{n+\kappa}$.

5.1.2 Privacy Security of COMET

In privacy attacks the adversary is concerned with distinguishing the COMET mode with an ideal authenticated encryption scheme. In addition to access of the encryption algorithm, the adversary is also allowed offline evaluations of the underlying block cipher. A trivial attack strategy is guessing the master key (as discussed in section 5.1.1). Non-trivially, the adversary can distinguish the modes from ideal if there is no randomness in some ciphertext (or tag) blocks. This is possible in the following two ways:

- Online-Online Block Matching: For a pair of distinct online (in this case encyrption) query block, the internal states matches. Then, the block that appears later will have non-random behavior. Note that this matching is only accidental and will happen with probability approx. $2^{-(n+\kappa)}$ in both COMET-128, and COMET-64. Thus it requires $D_eT \approx 2^{n+\kappa}$.

- Online-Offline Block Matching: This is similar to the state recovery attack strategy of section 5.1.1. Again this matching will happen accidentally with probability approx. $2^{-(n+\kappa)}$, which gives $D_eT \approx 2^{n+\kappa}$. To be a bit more precise, the product D_eT will be a little bit higher, approximately $2^{n+\kappa}/2n$. We ignore the logarithmic and constant factors as our security claims are well within the precise limits.

5.1.3 Integrity Security of COMET

In this case the adversary has to forge a fresh and valid ciphertext and tag pair. The adversary is allowed to make encryption queries to the encryption algorithm and forging queries to the decryption algorithm.

In forgery attack, the adversary can apply previous strategies of key or state recovery as in section 5.1.1. Previous strategies would lead to bounds of the form $T \approx 2^{\kappa}$ and $D_eT \approx 2^{n+\kappa}$. Some other attack strategies are described below:

- Tag Guessing: The adversary simply guesses random tag values for each forgery attempt. A tag guess will be valid with probability approx. 2^{-n}, which would give $D_v \approx 2^n$.

- Decryption Query Matching with Encryption Query Chain: Suppose the adversary gets the response $(c_1, c_2, c_3, c_4, c_5, t)$ for some encryption query, and then tries a decryption query of the form $(c'_1, c'_2, c'_3, c_4, c_5, t)$, of course, with a different nonce value. In this case the decryption query is valid if the next input internal state corresponding to c'_3 matches with the one corresponding to c_3. It can be shown that the probability of this event is bounded by approx. $nD_eD_v/2^{n+\kappa} + D_e/2^{2n}$, which gives $D_e \approx 2^n$ and $D_v \approx 2^{\kappa - \log_2 n}$.

- Decryption Query Matching with Offline Query Chain: This is similar to the previous case, in the sense, that the adversary tries to match a decryption query to a chain, except in this case the chain is constructed through offline queries. This would mean that the adversary constructed a chain of internal states for some fixed ciphertext and tag blocks using offline queries, and then matched a decryption query to this chain. Now the chain can be constructed by making block cipher queries in one of the two ways:

– Using forward only or backward only queries: This corresponds to the strategy where, the adversary makes, either E only or E^{-1} only queries. It can be shown that the probability of getting a successful forgery can be bounded by approx. $nD_vT/2^{n+\kappa} + T/2^{\kappa}$, which gives $T \approx 2^{\kappa}$ and $D_v \approx 2^n/n$.

– Using both forward and backward queries: This is the trickiest case, where the adversary makes both E and E^{-1} calls. The probability of getting a successful forgery, in this case, can be bounded by $2\sqrt{n}D_vT/2^{\kappa+n/2}$, which gives $D_v = 2^{\kappa+\frac{n}{2}-\frac{\log_2 n}{2}-1}/T$.

5.1.4 Validation of Security Claims

The security claims given in Table 3 follow from the rough lower bounds on D_e, D_v and T, as discussed in subsections 5.1.1-5.1.3. Importantly, it can be observed that the COMET mode of operation is secure, as long as the upper limits on D_e, D_v and T, as given in Table 3, are respected.

5.2 Security of Block Ciphers

All our submissions use COMET instantiated with well-known and fairly well-studied block ciphers AES-128/128, CHAM, and Speck-64/128. Some of the relevant cryptanalysis results on these block ciphers can be found, among others, in [10, 11, 12, 13, 14, 15, 16] for AES-128/128, [5] for CHAM, and [17, 18, 19, 20] for Speck-64/128.

We skip a detailed exposition on the cryptanalysis of these block ciphers, and instead summarize few standard cryptanalytic bounds. Since we use the ideal cipher view of the block ciphers, related-key attacks could be another relevant cryptanalytic appraoach. We remark here that in COMET-128, keys within a query are related by α-multiplication[1], which seems significantly harder then the usual XOR related-keys. In fact, to the best of our knowledge, this has not been analyzed till date.

5.2.1 Security of AES-128/128

The security of AES-128/128 is well-established in the community. To the best of our knowledge, the best single-key attack on AES-128/128 is the biclique attack by Bogdanov et al. [10], that recovers the key in approx. 2^{126} computations. Although there is a related-key attack on full-round AES-128/192 and AES-128/256, the same attack does not apply to AES-128/128, even in the usual XOR related-key setting, let alone the α-multiplication related-keys. In fact, [14] shows that AES-128/128 is almost as secure in related-key setting as it is in single-key setting. Recent distinguishers on AES-128/128 [11, 12, 13, 15], are applicable to round-reduced variants of AES-128/128, and hence not applicable in our case.

5.2.2 Security of CHAM

CHAM [5] has been designed to achieve high security in both single-key and related-key settings. The designers have claimed that the best possible attack strategy for CHAM-128/128 is the boomerang attack that breaks 47 rounds of CHAM-128/128, whereas the best attack strategy for CHAM-64/128 is the related-key boomerang attack that breaks 41 rounds of CHAM-64/128. Accordingly, full-round CHAM-128/128 and CHAM-64/128, each having 80 rounds, are expected to be secure against both single and related-key attacks. We could not find any third party analysis on the security of CHAM.

5.2.3 Security of Speck-64/128

The best single-key attack on Speck-64/128 [7] breaks 20 out of the 27 rounds with $2^{125.56}$ time and $2^{61.56}$ data complexity using differential cryptanalysis. In related-key settings, the rotational cryptanalysis of [19] of improves on the existing attacks for some versions of Speck, including Speck-32/64 and Speck-48/96, but it does not seem to improve the attacks on Speck-64/128. Overall, to the best of our knowledge there is no single-key or related-key attack on full-round Speck-64/128 with significantly better complexity than the brute-force approach.

5.3 Statement

We declare that there are no hidden weaknesses in the COMET mode of operation. Further, to the best of our knowledge, public third-party analysis do not raise any security threat to the submissions, COMET-128_AES-128/128, COMET-128_CHAM-128/128, COMET-128_CHAM-64/128 and COMET-128_Speck-64/128, within the data and time limit prescribed in Table 3 of section 4.

[1] Recall that α denotes the primitive element of the field.

6 Features and Limitations

Some of the standout features of our submissions are as follows:

1. Small State Size: Our main goal is to design AEAD schemes with minimum state size. COMET achieves minimal state size, in the sense that the only state it requires (apart from a constant number of bits) is used for the block cipher, i.e. $(n + \kappa)$-bit state. We believe that this is the smallest possible state size for nonce-based AEAD schemes with security level comparable with COMET.
2. Design Simplicity: The design of COMET is extremely simple. Apart from the block cipher evaluations, it only requires simple shift and XOR operations.
3. Efficiency: This point is closely related to the previous two points. As the design is nonce-based, we are able to keep it single pass, which makes the scheme quite efficient in both hardware and software. Apart from the block cipher call, only 1 shift and at most 2 XOR operations are required per block of AD or PT.

The only notable limitation of COMET seems to be the insecurity in nonce-misusing scenario. COMET can be broken in constant queries if the adversary can repeat the tweaks in arbitrary fashion. But we remark here that if the nonce is chosen uniformly at random then COMET-128 can still satisfy the NIST lightweight standardization requirements. Another, limitation is rekeying per block, which makes it difficult to precompute and store the round keys, thought this is not our motivation as we want to reduce the storage requirement.

Acknowledgments

Shay Gueron is a professor in the Department of Mathematics at the University of Haifa, Haifa, Israel and a Senior Principal Engineer at Amazon Web Services Inc., Seattle, USA.

Shay Gueron is a member of the Center for Cyber Law & Policy at the University of Haifa and a member of the BIU Center for Research in Applied Cryptography and Cyber Security.

Mridul Nandi is an associate professor in the Applied Statistics Unit at the Indian Statistical Institute, Kolkata, India.

Ashwin Jha is a PhD student in the Applied Statistics Unit at the Indian Statistical Institute, Kolkata, India.

Shay Gueron is supported by:

- the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Directorate in the Prime Minister's Office;
- the Israel Science Foundation (grant No. 1018/16);
- a grant from the Ministry of Science and Technology, Israel, and the Department of Science and Technology, Government of India;
- the Center for Cyber Law & Policy at the University of Haifa, in conjunction with the Israel National Cyber Directorate in the Prime Ministers Office.

Mridul Nandi is supported by:

- the DST/INT/ISR/P-20/2017 Indo-Israel Joint Research Programme by Department of Science and Technology, Government of India.

References

[1] Dworkin, M.: Recommendation for Block Cipher Modes of Operation – Methods and Techniques. NIST Special Publication 800-38A, National Institute of Standards and Technology, U. S. Department of Commerce (2001)

[2] Chakraborti, A., Datta, N., Nandi, M., Yasuda, K.: Beetle Family of Lightweight and Secure Authenticated Encryption Ciphers. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(2) (2018) 218–241

[3] Chakraborti, A., Datta, N., Nandi, M., Yasuda, K.: Beetle Family of Lightweight and Secure Authenticated Encryption Ciphers. IACR Cryptology ePrint Archive **2018** (2018) 805

[4] NIST: Announcing the ADVANCED ENCRYPTION STANDARD (AES). Fedral Information Processing Standards Publication FIPS 197, National Institute of Standards and Technology, U. S. Department of Commerce (2001)

[5] Koo, B., Roh, D., Kim, H., Jung, Y., Lee, D., Kwon, D.: CHAM: A family of lightweight block ciphers for resource-constrained devices. In: Information Security and Cryptology - ICISC 2017 - 20th International Conference, Seoul, South Korea, November 29 - December 1, 2017, Revised Selected Papers. (2017) 3–25

[6] Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The Simon and Speck Block Ciphers on AVR 8-Bit Microcontrollers. In: Lightweight Cryptography for Security and Privacy - Third International Workshop, LightSec 2014, Istanbul, Turkey, September 1-2, 2014, Revised Selected Papers. (2014) 3–20

[7] Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK Lightweight Block Ciphers. In: Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, June 7-11, 2015. (2015) 175:1–175:6

[8] Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: SIMON and SPECK: Block Ciphers for the Internet of Things. IACR Cryptology ePrint Archive **2015** (2015) 585

[9] Gueron, S., Lindell, Y.: Better bounds for block cipher modes of operation via nonce-based key derivation. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017. (2017) 1019–1036

[10] Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique cryptanalysis of the full AES. In: Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings. (2011) 344–371

[11] Grassi, L., Rechberger, C., Rønjom, S.: Subspace trail cryptanalysis and its applications to AES. IACR Trans. Symmetric Cryptol. **2016**(2) (2016) 192–225

[12] Grassi, L., Rechberger, C., Rønjom, S.: A new structural-differential property of 5-round AES. In: Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II. (2017) 289–317

[13] Rønjom, S., Bardeh, N.G., Helleseth, T.: Yoyo tricks with AES. In: Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I. (2017) 217–243

[14] Khoo, K., Lee, E., Peyrin, T., Sim, S.M.: Human-readable proof of the related-key security of AES-128. IACR Trans. Symmetric Cryptol. **2017**(2) (2017) 59–83

[15] Grassi, L.: Mixture differential cryptanalysis: a new approach to distinguishers and attacks on round-reduced AES. IACR Trans. Symmetric Cryptol. **2018**(2) (2018) 133–160

[16] Bar-On, A., Dunkelman, O., Keller, N., Ronen, E., Shamir, A.: Improved key recovery attacks on reduced-round AES with practical data and memory complexities. In: Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II. (2018) 185–212

[17] Biryukov, A., Roy, A., Velichkov, V.: Differential analysis of block ciphers SIMON and SPECK. In: Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers. (2014) 546–570

[18] Liu, Y., Fu, K., Wang, W., Sun, L., Wang, M.: Linear cryptanalysis of reduced-round SPECK. Inf. Process. Lett. **116**(3) (2016) 259–266

[19] Liu, Y., Witte, G.D., Ranea, A., Ashur, T.: Rotational-xor cryptanalysis of reduced-round SPECK. IACR Trans. Symmetric Cryptol. **2017**(3) (2017) 24–36

[20] Ashur, T., Bodden, D., Dunkelman, O.: Linear cryptanalysis using low-bias linear approximations. IACR Cryptology ePrint Archive **2017** (2017) 204