J Cryptol (2020) 33:703–741
https://doi.org/10.1007/s00145-019-09325-z

Journal of
CRYPTOLOGY

Blockcipher-Based Authenticated Encryption: How Small Can We Go?*

Avik Chakraborti
NTT Secure Platform Laboratories, Tokyo, Japan
chakraborti.avik@lab.ntt.co.jp

Tetsu Iwata
Nagoya University, Nagoya, Japan
tetsu.iwata@nagoya-u.jp

Kazuhiko Minematsu
NEC Corporation, Tokyo, Japan
k-minematsu@ah.jp.nec.com

Mridul Nandi
Applied Statistics Unit, Indian Statistical Institute, Kolkata, India
mridul.nandi@gmail.com

Communicated by François-Xavier Standaert.

Received 2 April 2018 / Revised 25 April 2019
Online publication 17 May 2019

Abstract. This paper presents a lightweight blockcipher-based authenticated encryption mode mainly focusing on minimizing the implementation size, i.e., hardware gates or working memory on software. The mode is called COFB, for COmbined FeedBack. COFB uses an n-bit blockcipher as the underlying primitive and relies on the use of a nonce for security. In addition to the state required for executing the underlying blockcipher, COFB needs only $n/2$ bits state as a mask. Till date, for all existing constructions in which masks have been applied, at least n bit masks have been used. Thus, we have shown the possibility of reducing the size of a mask without degrading the security level much. Moreover, it requires one blockcipher call to process one input block. We show COFB is provably secure up to $O(2^{n/2}/n)$ queries which is almost up to the standard birthday bound. We first present an idealized mode iCOFB along with the details of its provable security analysis. Next, we extend the construction to the practical mode COFB. We instantiate COFB with two 128-bit blockciphers, AES-128 and GIFT-128, and present their implementation results on FPGAs. We present two implementations, with and without CAESAR hardware API. When instantiated with AES-128 and implemented without CAESAR hardware API, COFB achieves only a few more than 1000 Look-Up-Tables (LUTs) while maintaining almost the same level of provable security

*A preliminary version of this paper was presented at CHES 2017 [25]

© International Association for Cryptologic Research 2019

as standard AES-based AE, such as GCM. When instantiated with GIFT-128, COFB performs much better in hardware area. It consumes less than 1000 LUTs while maintaining the same security level. However, when implemented with CAESAR hardware API, there are significant overheads both in hardware area and in throughput. COFB with AES-128 achieves about 1475 LUTs. COFB with GIFT-128 achieves a few more than 1000 LUTs. Though there are overheads, still both these figures show competitive implementation results compared to other authenticated encryption constructions.

Keywords. COFB, AES, GIFT, Authenticated encryption, Blockcipher.

1. Introduction

Authenticated encryption (AE) is a symmetric-key cryptographic primitive for providing both confidentiality and authenticity. Due to the recent rise in communication networks operated on small devices, the era of the so-called Internet of Things, AE is expected to play a key role in securing these networks.

In this paper, we study blockcipher modes for AE with primary focus on the hardware implementation size. Here, we consider the overhead in size; thus, the state memory size beyond the underlying blockcipher itself (including the key schedule) is the criteria we want to minimize, which is particularly relevant for hardware implementation. We observe this direction has not received much attention until the launch of CAESAR competition (see below), while it would be relevant for future communication devices requiring ultra low-power operations.

Generic Approaches One generic approach for reducing the implementation size of blockcipher modes is to use lightweight blockciphers. It covers a broad area of use cases, where standard AES is not suitable due to the implementation constraints, and one of the major criteria is area minimization. One of the most popular lightweight blockciphers is PRESENT [21] proposed in 2007. Since then, many have been proposed in the last decade, such as KATAN [24], LED [36], PICCOLO [62], PRINCE [23] and TWINE [63]. SIMON and SPECK [17] are proposed by NSA in 2014. More recent designs are SKINNY (which is a tweakable blockcipher [18]) and GIFT [15,16].

The other approach is to use standard AES implemented in a tiny, serialized core [51], where the latter is shown to be effective for various schemes including popular CCM [5] or OCB [43] modes, as shown in [22] and [13]. Still, this requires much larger number of clock cycles for each AES encryption than the standard round-based implementation and hence is not desirable when speed or energy is also a criteria in addition to size.

AE Modes with Small Memory CAESAR [3] is a competition for AE started in 2012. It attracted 57 AE schemes, and there are new schemes that were designed to minimize the implementation size while designed as a blockcipher mode (i.e., it uses a blockcipher as a black box). Among them, JAMBU [66] is considered to be one of the most relevant modes to our purpose, which can be implemented with $(1.5n+k)$-bit state memory, using n-bit blockcipher with k-bit key. However, the provable security result is not published for this scheme,[1] and the security claim about the confidentiality in the nonce misuse

[1] The authenticity result was briefly presented in the latest specification [66].

scenario was shown to be flawed [54]. We also point out that the rate of JAMBU is 1/2, i.e., it makes two blockcipher calls to process one input block. CLOC and SILC [39,40] have provable security results and were designed to minimize the implementation size; however, they have $(2n + k)$-bit state memory and the rate is also 1/2.

NIST Lightweight Cryptography Project Recently, the growing importance of lightweight applications has also been addressed by NIST's lightweight cryptography project [48], which recognizes the apparent lack of suitable AE standards to be used for lightweight applications. They highlighted the requirements under the backdrop of several arising applications like sensor networks, health care, distributed control systems and several others, where highly resource-constrained devices communicate among themselves.

A variant [14] of the COFB mode instantiated with the GIFT blockcipher has been submitted to the NIST lightweight crypto project [48]. This submission incorporates a longer nonce to be compatible with the submission guidelines and adopts a hardware optimized feedback function with lower XOR counts but with one bit authenticity degradation.

We next summarize our contributions.

A New Type of Feedback Function To reduce the state memory, it is natural to use feedback from the blocks involved in each blockcipher call, at the cost of losing parallelizability. There are existing feedback modes (such as ciphertext feedback of CBC encryption); however, we found that none of them is enough to fulfill our needs. We first formalize the feedback function as a linear function to take blockcipher output (Y) and plaintext block (M) to produce the corresponding ciphertext block (C) and the chain value as the next input to blockcipher (X). This formalization covers all previous popular feedback functions. Then, we propose a new type of feedback function, called *combined feedback*, where X is a linear function (not a simple XOR) of M and Y. We show that if the above linear function satisfies certain conditions, we could build a provably secure, small-state AE. We first present a mode of *tweakable random function* which has additional input called tweak in addition to n-bit block input, to demonstrate the effectiveness of combined feedback and intuition for provable security. The proposed scheme (iCOFB for idealized COmbined FeedBack) has a quite high provable security, comparable to ΘCB3 presented in the proof of OCB3 [43], and has small memory (n-bit block memory plus those needed for the primitive). In addition, it needs one primitive call to process n-bit message block.

Blockcipher AE mode with Combined Feedback Function Starting from iCOFB, we take a further step to propose a blockcipher mode using combined feedback. The main obstacle is the instantiation of tweakable random function (or, equivalently tweakable blockcipher [47]) using a blockcipher. We could use an existing tweakable blockcipher mode for this purpose, e.g., XEX [55] by Rogaway, and thanks to the standard birthday type security of XEX, the resulting blockcipher mode would also have standard birthday type security. However, the implementation of XEX or similar ones needs n-bit memory used as input mask to blockcipher, in addition to the main n-bit state block, implying $(2n + k)$-bit state memory. Therefore, instead of relying on the existing tweakable blockcipher modes, we instantiate the tweakable random function using only $n/2$-bit

Table 1. Comparison of AE modes, using an n-bit blockcipher with k-bit keys.

Scheme	State size	Rate	Parallel	Inverse-free	Sec. proof	Refs.
COFB	$1.5n + k$	1	No	Yes	Yes	This work
JAMBU	$1.5n + k$	$1/2$	No	Yes	Partial	[66]
CLOC/ SILC	$2n + k$	$1/2$	No	Yes	Yes	[39,40]
iFEED	$3n + k$	1	Only for Enc	Yes	Flawed [61]	[69]
OCB	$\geq 3n + k$	1	Yes	No	Yes	[43,55,56]

An inverse-free mode is a mode that does not need the blockcipher inverse (decryption) function both for encryption and for decryption. For JAMBU, the authenticity bound was briefly presented in [66]

mask and provide a dedicated security proof for our final proposal (mode), which we call COFB. We show COFB achieves almost birthday bound security, roughly up to $O(2^{n/2}/n)$ queries, based on the standard PRP assumption on the blockcipher.

COFB needs $n/2$-bit register for mask in addition to the registers required for holding round keys and the internal n-bit state for the blockcipher computation. Hence, the state size of COFB is $1.5n + k$ bits. The rate of COFB is 1, i.e., it makes one blockcipher call to process one input block, meaning it is as fast as encryption-only modes. On the downside, COFB is completely serial both for encryption and for decryption, which is inherent to the use of combined feedback. However, we argue that this is a reasonable trade-off, as tiny devices are our primal target platform for COFB. See Table 1 for comparison of COFB with others. The description and the security analysis of COFB in Sects. 4 and 5 have been described at the proceedings version of our paper in CHES 2017 [25].

Instantiations and Hardware Implementations We instantiate and implement COFB with the 128-bit version of the blockcipher AES known as AES-128. We also implement COFB with the 128-bit version of the blockcipher GIFT (described as GIFT-128 in [15,16]) to get an idea of the lightweight property of the COFB mode by checking how small (hardware area) it can go with a lightweight blockcipher. For the sake of completeness, we compare our implementation figures with various schemes (not limited to blockcipher modes) listed in the hardware benchmark framework called ATHENa [1]. The implementation details of COFB[AES] have already been described in [25,26]. COFB[AES] shows the impressive performance figures of COFB both for size and for speed compared to other AES-based AE modes. Moreover, if we implement COFB with GIFT, then it achieves much smaller area than COFB[AES] and is quite competitive to even ad hoc designs (see Sect. 6). The implementation details of COFB[GIFT] are also described in Sect. 6, which is a new contribution compared to [25]. For the sake of a fair benchmarking, we also provide CAESAR hardware API-supported hardware implementation results for both these two versions. Nevertheless, we think this comparison implies a good performance of COFB among others even using the standard AES-128 and implies COFB with a lightweight blockcipher to hit the limit of blockcipher-based AE's speed and size.

2. Preliminaries

Notation We fix a positive integer n which is the block size in bits of the underlying blockcipher E_K. Typically, we consider $n = 128$ and AES-128 [8] is the underlying blockcipher, where K is the 128-bit AES key. The empty string is denoted by λ. For any $X \in \{0, 1\}^*$, where $\{0, 1\}^*$ is the set of all finite bit strings (including λ), we denote the number of bits of X by $|X|$. Note that $|\lambda| = 0$. For two bit strings X and Y, $X\|Y$ denotes the concatenation of X and Y. A bit string X is called a *complete* (or *incomplete*) block if $|X| = n$ (or $|X| < n$, respectively). We write the set of all complete (or incomplete) blocks as $\mathcal{B}$ (or $\mathcal{B}^<$, respectively). Let $\mathcal{B}^\leq = \mathcal{B}^< \cup \mathcal{B}$ denote the set of all blocks. For $B \in \mathcal{B}^\leq$, we define $\overline{B}$ as follows:

$$\overline{B} = \begin{cases} 0^n & \text{if } B = \lambda \\ B\|10^{n-1-|B|} & \text{if } 0 < |B| < n \\ B & \text{if } |B| = n \end{cases}$$

Given non-empty $Z \in \{0, 1\}^*$, we define the parsing of Z into n-bit blocks as

$$(Z[1], Z[2], \ldots, Z[z]) \xleftarrow{n} Z, \qquad (1)$$

where $z = \lceil |Z|/n \rceil$, $|Z[i]| = n$ for all $i < z$ and $1 \leq |Z[z]| \leq n$ such that $Z = (Z[1] \,\|\, Z[2] \,\|\, \ldots \,\|\, Z[z])$. If $Z = \lambda$, we let $z = 1$ and $Z[1] = \lambda$. We write $||Z|| = z$ (number of blocks present in Z). We similarly write $(Z[1], Z[2], \ldots, Z[z]) \xleftarrow{m} Z$ to denote the parsing of the bit string Z into m-bit strings $Z[1], Z[2], \ldots, Z[z-1]$ and $1 \leq |Z[z]| \leq m$. Given any sequence $Z = (Z[1], \ldots, Z[s])$ and $1 \leq a \leq b \leq s$, we represent the sub sequence $(Z[a], \ldots, Z[b])$ by $Z[a \ldots b]$. For integers $a \leq b$, we write $[a \ldots b]$ for the set $\{a, a+1, \ldots, b\}$. For two bit strings X and Y with $|X| \geq |Y|$, we define the extended xor-operation as

$$X \underline{\oplus} Y = X[1 \ldots |Y|] \oplus Y \text{ and}$$
$$X \overline{\oplus}\, Y = X \oplus (Y\|0^{|X|-|Y|}),$$

where $(X[1], X[2], \ldots, X[x]) \xleftarrow{1} X$, and thus, $X[1 \ldots |Y|]$ denotes the first $|Y|$ bits of X. Also note that,

$$X \underline{\oplus} Y = (X \overline{\oplus} Y)[1 \ldots |Y|].$$

When $|X| = |Y|$, both operations reduce to the standard $X \oplus Y$.

Let $\gamma = (\gamma[1], \ldots, \gamma[s])$ be a tuple of equal-length strings. We define $\mathsf{mcoll}(\gamma) = r$ if there exist distinct $i_1, \ldots, i_r \in [1 \ldots s]$ such that $\gamma[i_1] = \cdots = \gamma[i_r]$ and r is the maximum of such integer. We say that $\{i_1, \ldots, i_r\}$ is an r-multi-collision set for γ. Finally, we use the notation E^c to denote the complement of an event E.

Authenticated Encryption and Security Definitions An authenticated encryption (AE) is an integrated scheme that provides both privacy of a plaintext $M \in \{0, 1\}^*$ and

authenticity of M as well as associated data $A \in \{0, 1\}^*$. Taking a nonce N (which is a value unique for each encryption) together with associated data A and plaintext M, the encryption function of AE, $\mathcal{E}_K$, produces a tagged ciphertext (C, T) where $|C| = |M|$ and $|T| = t$. Typically, t is fixed and we assume $n = t$ throughout the paper. The corresponding decryption function, $\mathcal{D}_K$, takes (N, A, C, T) and returns a decrypted plaintext M when the verification on (N, A, C, T) is successful, otherwise returns the atomic error symbol denoted by $\perp$.

Privacy Given an adversary $\mathcal{A}$, we define the *PRF advantage* of $\mathcal{A}$ against $\mathcal{E}$ as $\mathbf{Adv}^{\mathrm{prf}}_{\mathcal{E}}(\mathcal{A}) = |\Pr[\mathcal{A}^{\mathcal{E}_K} = 1] - \Pr[\mathcal{A}^{\$} = 1]|$, where \$ returns a random string of the same length as the output length of $\mathcal{E}_K$, by assuming that the output length of $\mathcal{E}_K$ is uniquely determined by the query. The PRF advantage of $\mathcal{E}$ is defined as

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathcal{E}}(q, \sigma, t) = \max_{\mathcal{A}} \mathbf{Adv}^{\mathrm{prf}}_{\mathcal{E}}(\mathcal{A}),$$

where the maximum is taken over all adversaries running in time t and making q queries with the total number of blocks in all the queries being at most σ. If $\mathcal{E}_K$ is an encryption function of AE, we call it the *privacy advantage* and write as $\mathbf{Adv}^{\mathrm{priv}}_{\mathcal{E}}(q, \sigma, t)$, as the maximum of all nonce-respecting adversaries (that is, the adversary can arbitrarily choose nonces provided all nonce values in the encryption queries are distinct).

Authenticity We say that an adversary $\mathcal{A}$ *forges* an AE scheme $(\mathcal{E}, \mathcal{D})$ if $\mathcal{A}$ is able to compute a tuple (N, A, C, T) satisfying $\mathcal{D}_K(N, A, C, T) \neq \perp$, without querying (N, A, M) for some M to $\mathcal{E}_K$ and receiving (C, T), i.e., (N, A, C, T) is a non-trivial forgery.

In general, a forger is nonce-respecting with respect to encryption queries, but can make q_f forging attempts without restriction on N in the decryption queries, that is, N can be repeated in the decryption queries and an encryption query and a decryption query can use the same N. The *forging advantage* for an adversary $\mathcal{A}$ is written as $\mathbf{Adv}^{\mathrm{auth}}_{\mathcal{E}}(\mathcal{A}) = \Pr[\mathcal{A}^{\mathcal{E}_K, \mathcal{D}_K} \text{ forges}]$, and we write

$$\mathbf{Adv}^{\mathrm{auth}}_{\mathcal{E}}((q_e, q_f), (\sigma_e, \sigma_f), t) = \max_{\mathcal{A}} \mathbf{Adv}^{\mathrm{auth}}_{\mathcal{E}}(\mathcal{A})$$

to denote the maximum forging advantage for all adversaries running in time t, making q_e encryption and q_f decryption queries with total number of queried blocks being at most σ_e and σ_f, respectively.

Unified Security Notion for AE The privacy and authenticity advantages can be unified into a single security notion as introduced in [34,57]. Let $\mathcal{A}$ be an adversary that only makes non-repeating queries to $\mathcal{D}_K$. Then, we define the AE advantage of $\mathcal{A}$ against $\mathcal{E}$ as

$$\mathbf{Adv}^{\mathrm{AE}}_{\mathcal{E}}(\mathcal{A}) = |\Pr[\mathcal{A}^{\mathcal{E}_K, \mathcal{D}_K} = 1] - \Pr[\mathcal{A}^{\$, \perp} = 1]|,$$

where the $\perp$-oracle always returns $\perp$ and the \$-oracle is as the privacy advantage. We similarly define $\mathbf{Adv}_{\mathcal{E}}^{\mathrm{AE}}((q_e, q_f), (\sigma_e, \sigma_f), t) = \max_{\mathcal{A}} \mathbf{Adv}_{\mathcal{E}}^{\mathrm{AE}}(\mathcal{A})$, where the maximum is taken over all adversaries running in time t, making q_e encryption and q_f decryption queries with the total number of blocks being at most σ_e and σ_f, respectively.

Blockcipher Security We use a blockcipher E as the underlying primitive, and we assume the security of E as a PRP (pseudorandom permutation). The *PRP advantage* of a blockcipher E is defined as $\mathbf{Adv}_E^{\mathrm{prp}}(\mathcal{A}) = |\Pr[\mathcal{A}^{E_K} = 1] - \Pr[\mathcal{A}^{\mathsf{P}} = 1]|$, where P is a random permutation uniformly distributed over all permutations over $\{0, 1\}^n$. We write

$$\mathbf{Adv}_E^{\mathrm{prp}}(q, t) = \max_{\mathcal{A}} \mathbf{Adv}_E^{\mathrm{prp}}(\mathcal{A}),$$

where the maximum is taken over all adversaries running in time t and making q queries. Here, σ does not appear as each query has a fixed length.

Coefficients-H Technique We outline the Coefficients-H technique developed by Patarin, which serves as a convenient tool for bounding the advantage (see [53,64]). We will use this technique (without giving a proof) to prove our main theorem. Consider two oracles $\mathcal{O}_0 = (\$, \perp)$ (the ideal oracle for the relaxed game) and $\mathcal{O}_1$ (real, i.e., our construction in the same relaxed game). Let $\mathcal{V}$ denote the set of all possible views an adversary can obtain. For any view $\tau \in \mathcal{V}$, we will denote the probability to realize the view as $\mathsf{ip}_{\mathsf{real}}(\tau)$ (or $\mathsf{ip}_{\mathsf{ideal}}(\tau)$) when it is interacting with the real (or ideal, respectively) oracle. We call these *interpolation probabilities*. Without loss of generality, we assume that the adversary is deterministic and fixed. Then, the probability space for the interpolation probabilities is uniquely determined by the underlying oracle. As we deal with stateless oracles, these probabilities are independent of the order of query responses in the view. Suppose we have a set of views, $\mathcal{V}_{\mathrm{good}} \subseteq \mathcal{V}$, which we call *good* views, and the following conditions hold:

1. In the game involving the ideal oracle $\mathcal{O}_0$ (and the fixed adversary), the probability of getting a view in $\mathcal{V}_{\mathrm{good}}$ is at least $1 - \epsilon_1$.
2. For any view $\tau \in \mathcal{V}_{\mathrm{good}}$, we have $\mathsf{ip}_{\mathsf{real}}(\tau) \geq (1 - \epsilon_2) \cdot \mathsf{ip}_{\mathsf{ideal}}(\tau)$.

Then, we have $|\Pr[\mathcal{A}^{\mathcal{O}_0} = 1] - \Pr[\mathcal{A}^{\mathcal{O}_1} = 1]| \leq \epsilon_1 + \epsilon_2$. The proof can be found at (say) [64].

3. Idealized Combined Feedback Mode

In this section, we introduce our idealized combined feedback mode. Let E_K be the underlying primitive, a blockcipher, with key K. Depending on how the next input block of E_K is determined from the previous output of E_K, a plaintext block, or a ciphertext block, we can categorize different types of feedback modes. Some of the feedback modes are illustrated in Fig. 1. The first three modes are known as the *message feedback mode*, *ciphertext feedback mode*, and *output feedback mode*, respectively. The examples using the first three modes can be found in the basic encryption schemes [4] or AE schemes [5,39,40,69]. The fourth mode, which uses additional (linear) operation

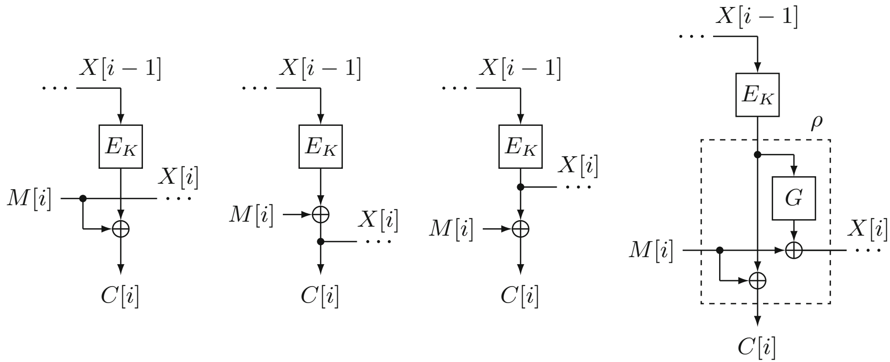

Fig. 1. Different types of feedback modes. We introduce the last feedback mode (called the combined feedback mode) in our construction.

$G : \mathcal{B} \to \mathcal{B}$, is new. We call it *combined feedback*. In the combined feedback mode, the next input block $X[i]$ of the underlying primitive E_K depends on at least two of the following three values: (1) previous output $E_K(X[i-1])$, (2) plaintext $M[i]$, and (3) ciphertext $C[i]$. With an appropriate choice of G, this feedback mode turns out to be useful for building small and efficient AE schemes. We provide a unified presentation of all types of feedback functions below.

Definition 1. *(Feedback Function)* A function $\rho : \mathcal{B} \times \mathcal{B} \to \mathcal{B} \times \mathcal{B}$ is called a feedback function (for an encryption) if there exists a function $\rho' : \mathcal{B} \times \mathcal{B} \to \mathcal{B} \times \mathcal{B}$ (used for decryption) such that

$$\forall Y, M \in \mathcal{B}, \quad \rho(Y, M) = (X, C) \Rightarrow \rho'(Y, C) = (X, M). \tag{2}$$

ρ is called a plaintext or output feedback if X depends only on M or Y, respectively (e.g., the first and third modes in Fig. 1). Similarly, it is called ciphertext feedback if X depends only on C in the function ρ' (e.g., the second mode in Fig. 1). All other feedback functions are called *combined feedback*.

The condition stated in Eq. (2) is sufficient for inverting the feedback computation from the ciphertext. Given the previous output block $Y = E_K(X[i-1])$ and a ciphertext block $C = C[i-1]$, we are able to compute $(X, M) = (X[i], M[i])$ by using $\rho'(Y, C)$.

In particular, when G is not the zero function nor the identity function, the combined feedback mode using this G is not reduced to the remaining three modes. It can be described as $\rho(Y, M) = (X, C) = (G(Y) \oplus M, Y \oplus M)$.

3.1. *iCOFB Construction*

The idealized version of our construction is described in Fig. 3 and illustrated in Fig. 2. Here we idealize in many ways from a real implementable AE construction. This is a simple warm up for the sake of simplicity and to understand the basic structure of our main construction. In the following construction, we simply assume that the last message

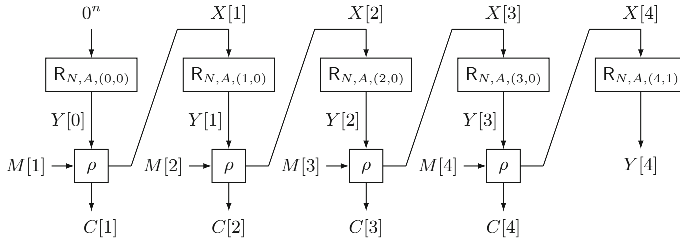

Fig. 2. iCOFB: It is based on a tweakable random function $\mathsf{R}_{N,A,(a,b)}$ and a feedback function ρ. The diagram shows how the tag and ciphertext computed for a three complete blocks message.

Algorithm iCOFB-$\mathcal{E}(N, A, M)$	**Algorithm** iCOFB-$\mathcal{D}(N, A, C, T)$
1. $(M[1], M[2], \ldots, M[m]) \xleftarrow{n} M$	1. $(C[1], C[2], \ldots, C[c]) \xleftarrow{n} C$
2. $t[0] \leftarrow (0, 0)$	2. $t[0] \leftarrow (0, 0)$
3. $Y[0] \leftarrow \mathsf{R}_{N,A,t[0]}(0^n)$	3. $Y[0] \leftarrow \mathsf{R}_{N,A,t[0]}(0^n)$
4. **for** $i = 1$ **to** m	4. **for** $i = 1$ **to** c
5. **if** $i < m$ **then** $t[i] \leftarrow (i, 0)$	5. **if** $i < c$ **then** $t[i] \leftarrow (i, 0)$
6. **else** $t[m] \leftarrow (m, 1)$	6. **else** $t[c] \leftarrow (c, 1)$
7. $(X[i], C[i]) \leftarrow \rho(Y[i-1], M[i])$	7. $(X[i], M[i]) \leftarrow \rho'(Y[i-1], C[i])$
8. $Y[i] \leftarrow \mathsf{R}_{N,A,t[i]}(X[i])$	8. $Y[i] \leftarrow \mathsf{R}_{N,A,t[i]}(X[i])$
9. $C \leftarrow (C[1], \ldots, C[m])$	9. $M \leftarrow (M[1], \ldots, M[c])$
10. $T \leftarrow Y[m]$	10. **if** $T = Y[c]$ **then return** M
11. **return** (C, T)	11. **else return** $\perp$

Fig. 3. Encryption and decryption algorithms of iCOFB AE mode. Here $M, C \in \mathcal{B}^+$ and $\rho, \rho' : \mathcal{B}^2 \to \mathcal{B}^2$. The choices of these functions are described in Sect. 3.2.

block is a complete block. In other words, all messages are elements of $\mathcal{B}^+ \stackrel{\text{def}}{=} \cup_{i \geq 1} \mathcal{B}^i$. We denote the set of all nonnegative integers as $\mathbb{Z}_{\geq 0}$. We also consider a tweakable random function R which takes tweak $(N, A, i, j) \in \mathcal{N} \times \{0, 1\}^* \times \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0}$ where N is called a nonce chosen from a nonce space $\mathcal{N}$, A is associated data, and the pair of nonnegative integers (i, j) is called a position tweak.

3.2. *The Feedback Function* ρ

It is easy to see that for a feedback function ρ, the decryption algorithm correctly decrypts a ciphertext. If we closely look into the correctness property, what we need that given (Y, C), the value of M should be uniquely computable. Once M is computed, X can be computed by applying ρ again. In this paper, we require very lightweight function, e.g., linear function, on the choice of ρ. If ρ is a linear function, then we can express ρ by a $2n \times 2n$ binary matrix

$$\begin{pmatrix} E_{1,1} & E_{1,2} \\ E_{2,1} & E_{2,2} \end{pmatrix}$$

where $E_{i,j}$'s are $n \times n$ binary matrices and the line 7 in the encryption algorithm of Fig. 3 becomes

$$X[i] = E_{1,1} \cdot Y[i-1] + E_{1,2} \cdot M[i],$$
$$C[i] = E_{2,1} \cdot Y[i-1] + E_{2,2} \cdot M[i].$$

We have the following lemma.

Lemma 1. *If ρ is a linear function satisfying Eq. (2), then $E_{2,2}$ must be invertible.*

Proof. If not, then there exist $M \neq M'$ with $E_{2,2} \cdot M = E_{2,2} \cdot M'$. Then, for any Y, $\rho(Y, M) = (X, C)$ and $\rho(Y, M') = (X', C)$. However, $\rho'(Y, C)$ cannot be both (X, M) and (X', M'). □

Assuming $E_{2,2}$ is invertible, let ρ be a linear feedback function satisfying Eq. (2). Then, ρ' can be chosen to be a linear function defined as follows:

$$(E_{1,1} + E_{1,2}E_{2,2}^{-1}E_{2,1}) \cdot Y[i-1] + E_{1,2} \cdot C[i] = X[i]$$
$$E_{2,2}^{-1}E_{2,1} \cdot Y[i-1] + E_{2,2}^{-1} \cdot C[i] = M[i].$$

We also express the above system of linear equations as

$$\begin{pmatrix} D_{1,1} & D_{1,2} \\ D_{2,1} & D_{2,2} \end{pmatrix} \cdot \begin{pmatrix} Y[i-1] \\ C[i] \end{pmatrix} = \begin{pmatrix} X[i] \\ M[i] \end{pmatrix}$$

where $D_{i,j}$'s are $n \times n$ matrix determined from the above linear equations. In particular, $D_{1,1} = (E_{1,1} + E_{1,2}E_{2,2}^{-1}E_{2,1})$, $D_{1,2} = E_{1,2}$, $D_{2,1} = E_{2,2}^{-1}E_{2,1}$, and $D_{2,2} = E_{2,2}^{-1}$.

Let $\boldsymbol{I}$ and $\boldsymbol{O}$ denote the identity matrix and zero matrix, respectively, of size n. We have seen that in all types of feedback modes, we define the ciphertext block C as $M \oplus Y$. They differ how the next input block X is defined. Let ρ_{OFB}, ρ_{CFB}, ρ_{PFB} denote the feedback functions for output, ciphertext, and plaintext feedback mode, respectively. Then, we have

$$\rho_{\mathsf{OFB}} = \begin{pmatrix} \boldsymbol{I} & \boldsymbol{O} \\ \boldsymbol{I} & \boldsymbol{I} \end{pmatrix}, \quad \rho_{\mathsf{CFB}} = \begin{pmatrix} \boldsymbol{I} & \boldsymbol{I} \\ \boldsymbol{I} & \boldsymbol{I} \end{pmatrix}, \quad \rho_{\mathsf{PFB}} = \begin{pmatrix} \boldsymbol{O} & \boldsymbol{I} \\ \boldsymbol{I} & \boldsymbol{I} \end{pmatrix}.$$

In this paper, we consider combined feedback function. By combined feedback, we mean that the following four matrices $E_{1,1}$, $E_{1,2}$, $D_{1,1}$ and $D_{1,2}$ are nonzero. Note that these matrices represent the effect of output vector and plaintext or ciphertext block to the next input block. In this paper, we fix our choice of ρ (and ρ') as

$$\rho = \begin{pmatrix} \boldsymbol{G} & \boldsymbol{I} \\ \boldsymbol{I} & \boldsymbol{I} \end{pmatrix}, \quad \rho' = \begin{pmatrix} \boldsymbol{I}+\boldsymbol{G} & \boldsymbol{I} \\ \boldsymbol{I} & \boldsymbol{I} \end{pmatrix}$$

where $\boldsymbol{G}$ is an invertible matrix such that $\boldsymbol{I} + \boldsymbol{G}$ is also invertible (see Fig. 1). We will specify one choice of $\boldsymbol{G}$ later.

3.3. *Security Analysis of the Idealized Construction*

In this section, we provide the security analysis of the idealized construction. Now we prove that under a very minimal assumption on ρ, the idealized version has perfect privacy and authenticity with negligible advantage. We say that a linear feedback function ρ is *valid*[2] (which is true for our choice of the feedback function) if

(P1) $E_{2,1}$ is invertible, **(A1)** $D_{1,2}$ is invertible, and **(A2)** $D_{1,1}$ is invertible,

where **(P1)** is needed for the privacy notion and **(A1)** and **(A2)** are needed for the authenticity notion. We remark that the invertibility of $E_{1,1}$ is not required.[3] Here, **A1** implies that for any two $C \neq C'$ and for any Y, $D_{1,1} \cdot Y + D_{1,2} \cdot C \neq D_{1,1} \cdot Y + D_{1,2} \cdot C'$. Note that we assume that $E_{2,2}^{-1}$ is invertible for correctness. Thus, **A2** means that the $2n \times 2n$ feedback matrix for ρ is also invertible. Another important implication of **A2** is the following:

$$\Pr[Y \xleftarrow{\$} \mathcal{B} : D_{1,1} \cdot Y + D_{1,2} \cdot C = X] = 2^{-n}, \ \forall (C, X) \in \mathcal{B}^2.$$

We have the following theorem.

Theorem 1. *If ρ is valid then for adversary $\mathcal{A}$ making q_e encryption queries and q_f forging attempts having at most ℓ_f many blocks, we have*

$$\boldsymbol{Adv}^{priv}_{\mathsf{iCOFB}}(\mathcal{A}) = 0, \quad \boldsymbol{Adv}^{auth}_{\mathsf{iCOFB}}(\mathcal{A}) \leq \frac{q_f(\ell_f + 1)}{2^n}.$$

Proof. We consider an adversary $\mathcal{A}$ which makes q_e nonce-respecting encryption queries (A_i, N_i, M_i) and receives (C_i, T_i), $1 \leq i \leq q_e$, and makes q_f decryption queries $(N_i^*, A_i^*, C_i^*, T_i^*)$, $1 \leq i \leq q_f$. The intermediate variables Z appeared in the both encryption and decryption algorithms are represented by $Z_i[j]$ for the jth computation of the ith query, where Z can be A, M, C, X, Y, and t (recall that t is a position tweak, i.e., the t is a function of the indices of the current data block). Note that T_i, T_i^* and N_i's are single blocks.

Perfect Privacy We prove the perfect privacy under the assumption that $E_{2,1}$ is invertible (i.e., **P1**). To show perfect privacy, it would be sufficient to show that $C_1, \ldots, C_{q_e}$ are uniformly and independently distributed and this would be true provided $Y_1, \ldots, Y_{q_e}$ are uniformly and independently distributed (due to **P1** which says that keeping all other fixed, influence from $Y_i[j]$ to $C_i[j]$ is bijective). Note that $Y_i[j] = \mathsf{R}_{N_i, A_i, t_i[j]}(X_i[j])$. We know that a tweakable random function returns a random string if the input concatenated with the tweak is fresh. So it is sufficient to show that for all i, j, $(N_i, A_i, t_i[j], X_i[j])$

[2] In addition to the following points, COFB needs invertibility of $E_{1,1}$'s, but this is not a mandatory option for iCOFB.

[3] While we need it for the security of COFB. This comes from the fact that the tweakable block cipher in COFB does not have the standard birthday security against CPAs because of its half mask.

is fresh. But this is easy to see as $\mathcal{A}$ is a nonce-respecting adversary, and for any i, the values of $t_i[j]$'s are distinct, and hence, $(N_i, t_i[j])$'s are distinct for all (i, j).

Authenticity Advantage We first consider the case of single forging attempt. Let (N^*, A^*, C^*, T^*) be the forging attempt, and let m^* be the length of C^* in n-bit blocks. We also define p as the length of the largest common prefix of $((C_\alpha[1], t_\alpha[1]), \ldots, (C_\alpha[m_1], t_\alpha[m_1]))$ and $((C^*[1], t^*[1]), \ldots, (C^*[m^*], t^*[m^*]))$ if $(N^*, A^*) = (N_\alpha, A_\alpha)$ for some $1 \le \alpha \le q$. Since all N_is are distinct, α is unique if exists. We define $p = 0$ if $(N^*, A^*) \ne (N_i, A_i)$ for all $1 \le i \le q$.

When $p > 0$, from the definition of tweak $t[\cdot]$, it is easy to see that $p < \min\{m_\alpha, m^*\}$. So, we have

$$Y_\alpha[p] = Y^*[p], \quad (C_\alpha[p+1], t_\alpha[p+1]) \ne (C^*[p+1], t^*[p+1]).$$

Claim. $(N^*, A^*, t^*[p+1], X^*[p+1])$ is fresh among all tweaked inputs.

Proof. (of Claim) We prove this in two sub-cases. We first note that for all $i \ne p+1$, $t_\alpha[i] \ne t^*[p+1]$ and so it would be sufficient to show that $(t_\alpha[p+1], X_\alpha[p+1]) \ne (t^*[p+1], X^*[p+1])$. If $C_\alpha[p+1] = C^*[p+1]$ then $X_\alpha[p+1] = X^*[p+1]$ but $t_\alpha[p+1] \ne t^*[p+1]$. Similarly, when $C_\alpha[p+1] \ne C^*[p+1]$, by **A1** condition, the next tweaked inputs are distinct. When $p = 0$, $N^* \ne N_i$ for all $1 \le i \le q$; hence, the claim holds. □

Therefore, $Y^*[p+1]$ is uniformly distributed given the values obtained so far, including the case $p = 0$. By **A2** condition, the probability of the next input also remains fresh with probability at least $(1 - 2^{-n})$. We can continue this until the last tweaked input, and so the last tweaked input remains fresh with probability at least $1 - (m^* - p)/2^n$, and if last tweaked input is fresh, the n-bit tag is completely random. Hence, the forging probability is $(m^* - p)/2^n + 1/2^n$. When $p = 0$, the first tweaked input $(N^*, A^*, t^*[0], 0^n)$ is fresh; hence, the forging probability is $(m^* + 1)/2^n$. Thus, the case $p = 0$ achieves the maximum forging probability $(m^* + 1)/2^n$ for a single attempt.

In the case of $q_f > 1$ forging attempts, the success probability is at most a multiplication of q_f and the bound for the single forging attempt. Hence, it is at most $q_f(\ell_f + 1)/2^n$ as we have $m^* \le \ell_f$ from the definition. This completes the proof. □

Now we see that **P1** and **A1** are also necessary. For example, if **P1** is not satisfied, then we find a nonzero block d such that, $d^{tr} \cdot E_{2,1} = 0^n$ where d^{tr} denotes the transposition of the vector. Then, for any Y, $d^{tr} \cdot E_{2,2} \cdot M = d^{tr} \cdot C$ where $C = E_{2,1} \cdot Y + E_{2,2} \cdot M$. This observation can be used as a privacy distinguisher.

Similarly if **A1** is not satisfied, then $D_{1,2}$ is not invertible. So there exists a nonzero d such that $D_{1,2} \cdot d = 0^n$. Thus, $D_{1,1} \cdot Y + D_{1,2} \cdot C^* = D_{1,1} \cdot Y + D_{1,2} \cdot C$ where $C^* = C + d$. This observation can be extended to an authenticity attack.

4. COFB: A Small-State, Rate-1, Inverse-Free AE Mode

In this section, we present our proposal, COFB, which has rate-1 (i.e., needs one blockcipher call for one input block), and is inverse-free, i.e., it does not need a blockcipher inverse (decryption). In addition to these features, this mode has a quite small-state size, namely $1.5n + k$ bits, in case '' the underlying blockcipher has an n-bit block and k-bit keys. We first specify the basic building blocks and parameters used in our construction.

4.1. *Specification*

Key and Blockcipher The underlying cryptographic primitive is an n-bit blockcipher, E_K. We assume that n is a multiple of 4. The key of the scheme is the key of the blockcipher, i.e., K.
Masking Function We define the masking function $\mathsf{mask} : \{0, 1\}^{n/2} \times \mathbb{N}^2 \rightarrow \{0, 1\}^{n/2}$ as follows:

$$\mathsf{mask}(\Delta, a, b) = \alpha^a \cdot (1 + \alpha)^b \cdot \Delta \tag{3}$$

We may write $\mathsf{mask}_\Delta(a, b)$ to mean $\mathsf{mask}(\Delta, a, b)$. Here, $\cdot$ denotes the multiplication over $\mathrm{GF}(2^{n/2})$, and α denotes the primitive element of the field. For the primitive polynomial defining the field, we choose the lexicographically first one, that is, $p(x) = \mathrm{x}^{64} + \mathrm{x}^4 + \mathrm{x}^3 + \mathrm{x} + 1$, following [6,38]. Rogaway [55] showed that for all $(a, b) \in \{0, \dots, 2^{51}\} \times \{0, \dots, 2^{10}\}$, the values of $\alpha^a \cdot (1+\alpha)^b$ are distinct. If we follow the notations of [55], the right- hand side of Eq. (3) could be written as $2^a 3^b \Delta$. For other values of n, we need to identify the primitive element α of the primitive polynomial and an integer L such that $\alpha^a \cdot (1+\alpha)^b$ are distinct for all $(a, b) \in \{0, \dots, L\} \times \{0, \dots, 4\}$. Then, the total allowed size of a message and associated data would be at most nL bits. We need this condition to prove the security claim. In particular, we have the following properties of the masking function.

Lemma 2. *For any $(a, b) \neq (a', b')$ chosen from the set $\{0, \dots, L\} \times \{0, \dots, 4\}$ (as described above), $c \in \{0, 1\}^{n/2}$ and a random $n/2$ bit string Δ, we have*

$$\Pr[\mathsf{mask}_\Delta(a, b) \oplus \mathsf{mask}_\Delta(a', b') = c] = \frac{1}{2^{n/2}}, \ \textit{and} \ \Pr[\mathsf{mask}_\Delta(a, b) = c] = \frac{1}{2^{n/2}}.$$

Proof of the first equation trivially follows from the fact that $\alpha^a \cdot (1+\alpha)^b$ are distinct for all $(a, b) \in \{0, \dots, L\} \times \{0, \dots, 4\}$.

Similar masking functions are frequently used in other modes, such as [10,49,55]; however, the masks are full n bits. The use of n-bit masking function usually allows to redefine the AE scheme as a mode of XE or XEX tweakable blockcipher [55], which significantly reduces the proof complexity. In our case, to reduce the state size, we decided to use the $n/2$-bit masking function, and as a result the proof is ad hoc and does not rely on XE or XEX.

Feedback Function Let $Y \in \{0,1\}^n$ and $(Y[1], Y[2], Y[3], Y[4]) \xleftarrow{n/4} Y$, where $Y[i] \in \{0,1\}^{n/4}$. We define $G : \mathcal{B} \to \mathcal{B}$ as $G(Y) = (Y[1] \oplus Y[4], Y[1], Y[2], Y[3])$.[4] We also view G as the $n \times n$ non-singular matrix, so we write $G(Y)$ and $G \cdot Y$ interchangeably. For $M \in \mathcal{B}^{\leq}$ and $Y \in \mathcal{B}$, we define $\rho_1(Y, M) = G \cdot Y \oplus \overline{M}$. The feedback function ρ and its corresponding ρ' are defined as

$$\rho(Y, M) = (\rho_1(Y, M), Y \underline{\oplus} M),$$
$$\rho'(Y, C) = (\rho_1(Y, Y \underline{\oplus} C), Y \underline{\oplus} C).$$

Note that when $(X, M) = \rho'(Y, C)$, $X = (G \oplus I) \cdot Y \underline{\oplus} C$. Our choice of G ensures that $I \oplus G$ is also invertible matrix. So when Y is chosen randomly for both computations of X (through ρ and ρ'), X also behaves randomly. We need this property when we bound probability of bad events later.

Furthermore, in order to handle the case that the last ciphertext block is shorter than n bits, we need more conditions. That is, for the last ciphertext block ($C[m]$ with $|C[m]| = b \leq n$), the state S is updated by ρ as

$$S' = G(S) + \mathsf{ozp}(\mathsf{msb}_b(S) + C[m]) = G(S) + (\mathsf{msb}_b(S) \,\|\, 0 \ldots 0) + \mathsf{ozp}(C[m]).$$

To make sure that S' is random whenever S is random (otherwise the tag may have smaller entropy), we also need the following condition:

Condition $G + M_{\mathsf{msb}[i]}$ is regular (i.e., binary matrix of rank n) for any $i = 1, 2, \ldots, n$, where $M_{\mathsf{msb}[i]}$ denotes the matrix for extracting the first i bits: $M_{\mathsf{msb}[i]} \cdot X = \mathsf{msb}_i(X)$.

Since $M_{\mathsf{msb}[n]} = I$, this also covers the case $G + I$.[5]

The matrix G of the current specification corresponds to

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

It is also a transpose of a matrix called $M14$ in [39] and fulfills the condition mentioned above.

Tweak Value for The Last Block Given $B \in \{0,1\}^*$, we define $\delta_B \in \{1, 2\}$ as follows:

$$\delta_B = \begin{cases} 1 & \text{if } B \neq \lambda \text{ and } n \text{ divides } |B| \\ 2 & \text{otherwise.} \end{cases} \tag{4}$$

[4] We updated the definition of the feedback function.

[5] The G function in the previous version [25] does not have the maximum rank. More specifically, $G + M_{\mathsf{msb}[3n/4]}$ has rank $3n/4$ which is the lowest among all the cases.

This will be used to differentiate the cases that the last block of B is n bits or shorter, for B being associated data or plaintext or ciphertext. We also define a formatting function Fmt for a pair of bit strings (A, Z), where A is associated data and Z could be either a plaintext or a ciphertext. Let $(A[1], \ldots, A[a]) \xleftarrow{n} A$ and $(Z[1], \ldots, Z[z]) \xleftarrow{n} Z$. We define $\mathsf{t}[i]$ as follows:

$$\mathsf{t}[i] = \begin{cases} (i, 0) & \text{if } i < a \\ (a-1, \delta_A) & \text{if } i = a \\ (i-1, \delta_A) & \text{if } a < i < a+z \\ (a+z-2, \delta_A + \delta_Z) & \text{if } i = a+z \end{cases}$$

Now, the formatting function $\mathsf{Fmt}(A, Z)$ returns the following sequence:

$$\big((A[1], \mathsf{t}[1]), \ldots, (\overline{A[a]}, \mathsf{t}[a]), (Z[1], \mathsf{t}[a+1]), \ldots, (\overline{Z[z]}, \mathsf{t}[a+z])\big),$$

where the first coordinate of each pair specifies the input block to be processed, and the second coordinate specifies the exponents of α and $1+\alpha$ to determine the constant over $\mathrm{GF}(2^{n/2})$. Let $\mathbb{Z}_{\geq 0}$ be the set of nonnegative integers and $\mathcal{X}$ be some non-empty set. We say that a function $f : \mathcal{X} \rightarrow (\mathcal{B} \times \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0})^+$ is *prefix-free* if for all $X \neq X'$, $f(X) = (Y[1], \ldots, Y[\ell])$ is not a prefix of $f(X') = (Y'[1], \ldots, Y'[\ell'])$ (in other words, $(Y[1], \ldots, Y[\ell]) \neq (Y'[1], \ldots, Y'[\ell])$). Here, for a set $\mathcal{S}$, $\mathcal{S}^+$ means $\mathcal{S} \cup \mathcal{S}^2 \cup \cdots$, and we have the following lemma.

Lemma 3. *The function* $\mathsf{Fmt}(\cdot)$ *is prefix-free.*

The proof is more or less straightforward, and hence, we skip it.

We present the specifications of COFB in Fig. 5, where α and $(1+\alpha)$ in Eq. (3) are written as 2 and 3. See also Fig. 4. The encryption and decryption algorithms are denoted by COFB-$\mathcal{E}_K$ and COFB-$\mathcal{D}_K$. We remark that the nonce length is $n/2$ bits, which is enough for the security up to the birthday bound. The nonce is processed as $E_K(0^{n/2} \| N)$ to yield the first internal chaining value. The encryption algorithm takes non-empty A and non-empty M, and outputs C and T such that $|C| = |M|$ and $|T| = n$. The decryption algorithm takes (N, A, C, T) with $|A|, |C| \neq 0$ and outputs M or $\perp$. Note that some of building blocks described above are not presented in Fig. 5, since they are introduced for the proof. An equivalent presentation using them is presented in Fig. 6.

5. Security of COFB

We present the security analysis of COFB in Theorem 2. Before going to the proof, as mentioned earlier, we would like to mention that we use the function Fmt and Lemma 3 in the proof to make it easy to understand. We would also like to mention that we instantiate iCOFB with COFB by choosing

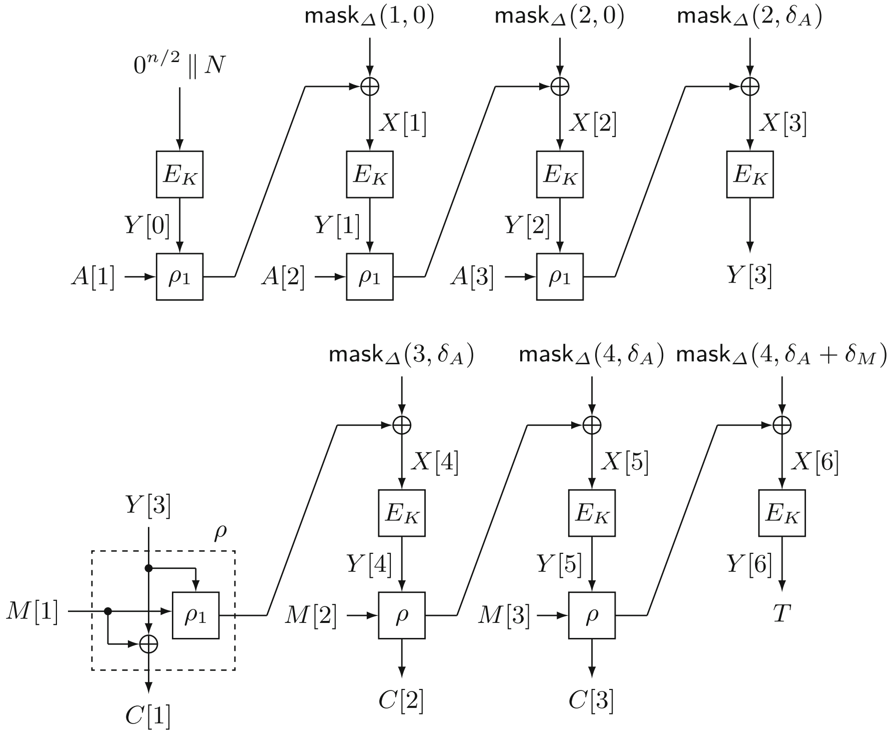

Fig. 4. Encryption of COFB for 3-block associated data and plaintext.

$$\mathsf{R}_{N,A,(i,j)}(X) = \begin{cases} f(N,A) & \text{cases if } i=0,\ j=0 \\ E_K(X \oplus \mathsf{mask}_\Delta(a+i-1,\delta_A)) & \text{cases if } i<m,\ j=0 \\ E_K(X \oplus \mathsf{mask}_\Delta(a+m-2,\delta_A+\delta_M)) & \text{cases if } i=m,\ j=1 \end{cases}$$

where $f(N, A)$ is the function that simulates the associated data phase and outputs $Y[a]$ (Line 1–11, Fig. 5, K is implicit and chosen uniformly from the key space and $X = 0^n$ in this case). Δ (computed using E_K and N), a, m, δ_A and δ_M are described as in the previous section, and we instantiate ρ by the feedback function described in the previous section. However, the security proof of COFB does not follow from that of iCOFB, since as a tweakable PRF, the security of R is only guaranteed up to $n/4$ bits, and thus, we cannot rely on the hybrid argument to show the security of COFB. We next proceed with our proof for our instantiation.

Theorem 2. (Main theorem)

$$\boldsymbol{Adv}^{AE}_{\mathsf{COFB}}((q_e,q_f),(\sigma_e,\sigma_f),t) \le \boldsymbol{Adv}^{prp}_{AES}(q',t') + \frac{0.5(q')^2}{2^n} + \frac{4\sigma_e}{2^{n/2}} + \frac{(q_e+\sigma_e+2\sigma_f)\cdot\sigma_f + q_f}{2^n}.$$

Algorithm Mask-Gen(K, N)

1. $Y[0] \leftarrow E_K(0^{n/2} \parallel N)$
2. $(Y^1[0], \ldots, Y^4[0]) \xleftarrow{n/4} Y[0]$
3. $\Delta \leftarrow Y^2[0] \parallel Y^3[0]$
4. **return** $(\Delta, Y[0])$

Algorithm $\mathsf{COFB}\text{-}\mathcal{E}_K(N, A, M)$

1. $(\Delta, Y[0]) \leftarrow$ Mask-Gen(K, N)
2. $(A[1], \ldots, A[a]) \xleftarrow{n} A$
3. $(M[1], \ldots, M[m]) \xleftarrow{n} M$
4. **for** $i = 1$ **to** $a - 1$
5. $\quad \Delta \leftarrow 2\Delta$
6. $\quad X[i] \leftarrow (A[i] \oplus G \cdot Y[i-1]) \overline{\oplus} \Delta$
7. $\quad Y[i] \leftarrow E_K(X[i])$
8. **if** $|A[a]| = n$ **then** $\Delta \leftarrow 3\Delta$
9. **else** $\Delta \leftarrow 3^2\Delta$
10. $X[a] \leftarrow (\overline{A[a]} \oplus G \cdot Y[a-1]) \overline{\oplus} \Delta$
11. $Y[a] \leftarrow E_K(X[a])$
12. **for** $i = 1$ **to** $m - 1$
13. $\quad X[i+a] \leftarrow (M[i] \oplus G \cdot Y[i+a-1]) \overline{\oplus} \Delta$
14. $\quad Y[i+a] \leftarrow E_K(X[i+a])$
15. $\quad C[i] \leftarrow Y[i+a-1] \oplus M[i]$
16. $\quad$ **if** $i < m - 1$ **then** $\Delta \leftarrow 2\Delta$
17. **if** $|M[m]| = n$ **then** $\Delta \leftarrow 3\Delta$
18. **else** $\Delta \leftarrow 3^2\Delta$
19. $X[a+m] \leftarrow (\overline{M[m]} \oplus G \cdot Y[a+m-1]) \overline{\oplus} \Delta$
20. $C[m] \leftarrow Y[a+m-1] \underline{\oplus} M[m]$
21. $T \leftarrow E_K(X[a+m])$
22. **return** (C, T)

Algorithm $\mathsf{COFB}\text{-}\mathcal{D}_K(N, A, C, T)$

1. $(\Delta, Y[0]) \leftarrow$ Mask-Gen(K, N)
2. $(A[1], \ldots, A[a]) \xleftarrow{n} A$
3. $(C[1], \ldots, C[c]) \xleftarrow{n} C$
4. **for** $i = 1$ **to** $a - 1$
5. $\quad \Delta \leftarrow 2\Delta$
6. $\quad X[i] \leftarrow (A[i] \oplus G \cdot Y[i-1]) \overline{\oplus} \Delta$
7. $\quad Y[i] \leftarrow E_K(X[i])$
8. **if** $|A[a]| = n$ **then** $\Delta \leftarrow 3\Delta$
9. **else** $\Delta \leftarrow 3^2\Delta$
10. $X[a] \leftarrow (\overline{A[a]} \oplus G \cdot Y[a-1]) \overline{\oplus} \Delta$
11. $Y[a] \leftarrow E_K(X[a])$
12. **for** $i = 1$ **to** $c - 1$
13. $\quad X[i+a] \leftarrow (C[i] \oplus Y[i+a-1] \oplus G \cdot Y[i+a-1]) \overline{\oplus} \Delta$
14. $\quad M[i] \leftarrow Y[i+a-1] \oplus C[i]$
15. $\quad Y[i+a] \leftarrow E_K(X[i+a])$
16. $\quad$ **if** $i < c - 1$ **then** $\Delta \leftarrow 2\Delta$
17. **if** $|C[a]| = n$ **then** $\Delta \leftarrow 3\Delta$
18. **else** $\Delta \leftarrow 3^2\Delta$
19. $X[a+c] \leftarrow (\overline{C[c]} \oplus Y[a+c-1] \oplus G \cdot Y[a+c-1]) \overline{\oplus} \Delta$
20. $M[c] \leftarrow Y[a+c-1] \underline{\oplus} C[c]$
21. $T' \leftarrow E_K(X[a+c])$
22. $M \leftarrow (M[1], \ldots, M[c])$
23. **if** $T' = T$ **then return** M
24. **else return** $\perp$

Fig. 5. Encryption and decryption algorithms of COFB.

Module Mask-Gen(K, N)

1. $Y[0] \leftarrow E_K(0^{n/2} \parallel N)$
2. $(Y^1[0], \ldots, Y^4[0]) \xleftarrow{n/4} Y[0]$
3. $\Delta \leftarrow Y^2[0] \| Y^3[0]$
4. **return** $(\Delta, Y[0])$

Algorithm $\mathsf{COFB}\text{-}\mathcal{E}_K(N, A, M)$

1. $(\Delta, Y[0]) \leftarrow$ Mask-Gen(K, N)
2. $(A[1], \ldots, A[a]) \xleftarrow{n} A$
3. $(M[1], \ldots, M[m]) \xleftarrow{n} M$
4. $\ell \leftarrow a + m$
5. $((B[1], \mathsf{t}[1]), \ldots, (B[\ell], \mathsf{t}[\ell])) \leftarrow \mathsf{Fmt}(A, M)$
6. **for** $i = 1$ **to** ℓ
7. $\quad X[i] \leftarrow (B[i] \oplus G \cdot Y[i-1]) \, \overline{\oplus} \, \mathsf{mask}_\Delta(\mathsf{t}[i])$
8. $\quad Y[i] \leftarrow E_K(X[i])$
9. $\quad$ **if** $i > a$ **then**
10. $\quad\quad C[i-a] \leftarrow Y[i-1] \, \underline{\oplus} \, M[i-a]$
11. $T \leftarrow Y[\ell]$
12. **return** (C, T)

Algorithm $\mathsf{COFB}\text{-}\mathcal{D}_K(N, A, C, T)$

1. $(\Delta, Y[0]) \leftarrow$ Mask-Gen(K, N)
2. $(A[1], \ldots, A[a]) \xleftarrow{n} A$
3. $(C[1], \ldots, C[c]) \xleftarrow{n} C$
4. $\ell \leftarrow a + c$
5. $((B[1], \mathsf{t}[1]), \ldots, (B[\ell], \mathsf{t}[\ell])) \leftarrow \mathsf{Fmt}(A, C)$
6. **for** $i = 1$ **to** ℓ
7. $\quad$ **if** $i \leq a$ **then**
8. $\quad\quad X[i] \leftarrow (B[i] \oplus G \cdot Y[i-1]) \overline{\oplus} \mathsf{mask}_\Delta(\mathsf{t}[i])$
9. $\quad$ **else** $X[i] \leftarrow (B[i] \oplus Y[i-1] \oplus G \cdot Y[i-1])$ $\overline{\oplus}$ $\mathsf{mask}_\Delta(\mathsf{t}[i])$
10. $\quad Y[i] \leftarrow E_K(X[i])$
11. **for** $i = 1$ **to** c
12. $\quad M[i] \leftarrow Y[i+a-1] \, \underline{\oplus} \, C[i]$
13. $M \leftarrow (M[1], \ldots, M[c])$
14. $T' \leftarrow Y[\ell]$
15. **if** $T' = T$ **then return** M
16. **else return** $\perp$

Fig. 6. A presentation of COFB using Fmt function. This is equivalent to Fig. 5.

where $q' = q_e + q_f + \sigma_e + \sigma_f$, which corresponds to the total number of blockcipher calls through the game, and $t' = t + O(q')$. To be precise, σ_e is the total number of blocks in q_e encryption queries and σ_f is the total number blocks in the q_f decryption queries.

Proof. Fix a **deterministic non-repeating** query making distinguisher adv that interacts with either the

1. Real oracle (COFB-$\mathcal{E}_K$) or the
2. Ideal oracle (\$)

making **exactly** q_e encryption queries $(N_i, A_i, M_i), i = 1 \ldots q_e$ with total σ_e many blocks and tries to forge with **exactly** q_f many queries $(N_i^*, A_i^*, C_i^*, T_i^*), i = 1 \ldots q_f$ having a total of σ_f many blocks. We also assume that adversary always makes fresh forging attempt. More formally, it cannot submit (N_i, A_i, C_i, T_i) for some i, as a forging attempt.

Let M_i and A_i have m_i and a_i blocks, respectively, and C_i^* and A_i^* have c_i^* and a_i^* blocks, respectively, for every i. We use X and Y to denote the intermediate variables (blockcipher input outputs) corresponding to the encryption queries. We also use X^* and Y^* to denote the intermediate variables (blockcipher input outputs) corresponding to the forging queries. Note that some of the inputs and outputs for forging queries can be determined by those of encryption queries.

Without loss of generality, we can assume $q' := q_e + q_f + \sigma_e + \sigma_f \le 2^{\frac{n}{2}-1}$ since otherwise, the bound obviously holds as the right hand side becomes more than one.

We use the notation X, Y etc. (mathsf notations) to denote random variables and the capital letters X, Y etc. to denote fixed values. Note that all values appeared in the transcript or internally are random variables due to randomness either from the real world or from the ideal world.

5.1. *Hybrid Argument (Transition from COFB-$\mathcal{E}_K$ to COFB-R)*

The first transition we make is to use an n-bit (uniform) random permutation P instead of E_K and then to use an n-bit (uniform) random function R instead of P. This two-step transition requires the first two terms of our bound, from the standard PRP–PRF switching lemma and from the computation to the information security reduction (e.g., see [19]). Then what we need is a bound for COFB using R, denoted by COFB-R. Let COFB^{-1}-R be the corresponding decryption function. So it is sufficient to prove

$$\mathbf{Adv}^{\mathrm{AE}}_{\mathsf{COFB\text{-}R}}((q_e, q_f), (\sigma_e, \sigma_f), \infty) \le \frac{4\sigma_e}{2^{n/2}} + \frac{(q_e + \sigma_e + 2\sigma_f) \cdot \sigma_f + q_f}{2^n}. \quad (5)$$

5.2. *Description of the Real Oracle*

The Real oracle simulates COFB-R to adv honestly. In case of q_e encryption queries, for $i = 1, \ldots, q_e$, we write (N_i, A_i, M_i) and (C_i, T_i) to denote the ith encryption query and response, such that

$$(C_i, T_i) = \mathsf{COFB\text{-}R}(N_i, A_i, M_i).$$

Here,

$$A_i = (A_i[1], \ldots, A_i[a_i]), M_i = (M_i[1], \ldots, M_i[m_i]), C_i = (C_i[1], \ldots, C_i[m_i]).$$

Let $\ell_i = a_i + m_i$, which denotes the total input block length for the ith encryption query. We write $X_i[j]$ (resp. $Y_i[j]$) for $i = 1, \ldots, q_e$ and $j = 0, \ldots, \ell_i$ to denote the jth input (resp. output) of the internal R invoked at the ith encryption query, where the order of invocation follows the specification shown in Fig. 5. We remark that $X_i[0] = 0^{n/2} \| N_i$ and $Y_i[\ell_i] = T_i$ for all $i = 1, \ldots, q_e$. Similarly, we write Δ_i to denote $Y_i^2[0] \| Y_i^3[0]$ where $Y_i^1[0] \| \cdots \| Y_i^4[0] \xleftarrow{n/4} Y_i[0]$. Note that, total number input blocks queried to the encryption oracle (corresponding to the q_e queries) is σ_e.

In case of all the q_f forge queries, for $i' = 1, \ldots, q_f$, we write $(N_{i'}^*, A_{i'}^*, C_{i'}^*, T_{i'}^*)$ and $Z_{i'}^*$ to denote the i'th forging attempt and response, such that

$$Z_{i'}^* = \mathsf{COFB}^{-1}\text{-}\mathsf{R}(N_{i'}^*, A_{i'}^*, C_{i'}^*, T_{i'}^*),$$

where $Z_{i'}^*$ is a *valid* message $M_{i'}^* = (M_{i'}^*[1], \ldots, M_{i'}^*[c_{i'}^*])$ or $\perp$. Here, $A_{i'}^* = (A_{i'}^*[1], \ldots, A_{i'}^*[a_{i'}^*])$ and $C_{i'}^* = (C_{i'}^*[1], \ldots, C_{i'}^*[c_{i'}^*])$. Let $\ell_{i'}^* = a_{i'}^* + c_{i'}^*$, which denotes the total input block length for the i'th forge query. We write $X_{i'}^*[j']$ (resp. $Y_{i'}^*[j']$) for $i' = 1, \ldots, q_f$ and $j = 0, \ldots, \ell_i^*$ to denote the jth input (resp. output) of the internal R invoked at the ith forging attempt, where the order of invocation follows the specification shown in Fig. 5.

Definition 2. For any i, let p_i denote the length of the longest common prefix of $\mathsf{Fmt}(A_i^*, C_i^*)$ and $\mathsf{Fmt}(A_j, C_j)$ where $N_j = N_i^*$. Note that there cannot be more than one j as the adversary is nonce respecting. If there is no such j, we define $p_i = -1$.

In the above definition, it holds that $p_i < \min\{\ell_i^*, \ell_j\}$ as Fmt is prefix-free. So, $X_i^*[0 \ldots p_i]$ is same as $X_j[0 \ldots p_i]$ due to common prefix. Moreover, $X_i^*[p_i + 1]$ is also determined as $X_i^*[p_i + 1] = M_i^*[p_i - a_i^* + 1] \oplus G \cdot Y^*[p_i] \oplus \mathsf{mask}_{\Delta_j}(t_j[p_i])$.

5.3. *Releasing More Information*

We release more information to the adversary, which can only gain the advantage. First, after completing all queries and forging attempts (i.e., decryption queries), let the adversary adv learn all the Y-values for all encryption queries only (which also allows adversary to learn X-values too). In addition, we also release the Y^*-values (consequently X^*-values) for the decryption queries (they are sampled according to the Y-values from the encryption queries). To be precise, we sample Y^* by the following way. First we denote the set of all prefixes of the formatted inputs for all the q_e encryption queries by P_{enc} (see the following example). We similarly denote the set of all formatted input prefixes for all the q_f decryption queries by P_{dec}.

Example 1. Let $q_e = 2$ and $((N, t[1]), (A, t[2]), (M, t[3])) \leftarrow \mathsf{Fmt}(N, A, M)$ and $((N', t'[1]), (A', t'[2]), (M', t'[3])) \leftarrow \mathsf{Fmt}(N', A', M')$ with $N \neq N'$ be the two

encryption queries. Hence,

$$P_{enc} = \{(N, t[1]), (N', t'[1]), ((N, t[1]), (A, t[2])), ((N, t[1]), (A, t[2]), (M, t[3])), \\ ((N', t'[1]), (A', t'[2])), ((N', t'[1]), (A', t'[2]), (M', t'[3]))\}. \qquad \square$$

Note that, $|P_{enc}| = \Sigma_{i=1}^{q_e}(\ell_i + 1)$ as nonces do not repeat. However, $|P_{dec}| \le \Sigma_{i=1}^{q_f}(\ell_i^* + 1)$ as nonces can repeat in the decryption queries. For all $x \in P_{dec}$, we sample Z_x uniformly from $\{0, 1\}^n$. Let

$$(Z_i^*[1], \ldots, Z_i^*[\ell_i^*]) \leftarrow (A_i^*[1], \ldots, A_i^*[a_i^*], C_i^*[1], \ldots C_i^*[c_i^*]).$$

Note that, the prefix set corresponding to $Y_i^*[j]$ denoted by $\mathsf{prefix}(Y^*, i, j)$ is

$$((N_i^*, *), (Z_i^*[1], *), \ldots, (Z_i^*[j], *)),$$

where $*$ denotes certain t values. Similarly, the prefix set denoted by $\mathsf{prefix}(Y, i, j)$ corresponding to $Y_i[j]$ is

$$((N_i, *), (Z_i[1], *), \ldots, (Z_i[j], *)),$$

where

$$(Z_i[1], \ldots, Z_i[\ell_i^*]) \leftarrow (A_i[1], \ldots, A_i[a_i^*], M_i[1], \ldots M_i[m_i]).$$

We now sample Y^* values by the following way

$$\mathsf{Y}_i^*[j] = \begin{cases} \mathsf{Y}_{i'}[j'] & \text{if } j \le p_i \text{ and } \mathsf{prefix}(\mathsf{Y}^*, i, j) = \mathsf{prefix}(\mathsf{Y}, i', j') \\ \mathsf{Z}_x & \text{if } j > p_i, \text{ and } \mathsf{prefix}(\mathsf{Y}^*, i, j) = x. \end{cases}$$

Note that, X^*s are determined from the Y^*s and C^*s.

5.4. *Description of the* Ideal *Oracle*

The original ideal oracle returns all ciphertext C_i and T_i randomly for encryption queries. In case of all the q_f forge queries $(N_{i'}^*, A_{i'}^*, C_{i'}^*, T_{i'}^*), i' \in \{1, \ldots, q_f\}$, the oracle returns $\perp$ (here we assume that the adversary makes only fresh queries). The encryption of the ideal oracle can equivalently be sampled in the following way:

We first sample all $Y_i[j]$ blocks uniformly and independently from $\{0, 1\}^n$, $\forall i \in \{0, \ldots, q_e\}, \forall j \in \{0, \ldots, \ell_i\}$. The ciphertext blocks during the message processing phase are computed as

$$C_i[j - a_i] = M_i[j - a_i] \oplus Y_i[j], \forall i \in \{1, \ldots q_e\}, j \in \{a_i, \ldots, \ell_i - 2\}.$$

Note that, $\forall i \in \{1, \ldots, q_e\}$, Δ_i values are computed as $Y_i^2[0] \| Y_i^3[0]$. Also, $\forall i \in \{1, \ldots q_e\}$, $C_i[m_i]$ is computed as

$$C_i[m_i] = M_i[m_i] \underline{\oplus} Y_i[\ell_i - 1].$$

Note that the last message block of the ith encryption query $M_i[m_i]$ may not be full (i.e, $|M_i[m_i]| < n$). The tag T_i, $\forall i \in \{1, \ldots, q_e\}$ is computed as

$$T_i = Y_i[\ell_i].$$

As ciphertext is computed xoring message blocks by uniformly sampled Y-values, the equivalent ideal oracle actually samples C_i randomly. The same is true for tag values.

5.5. *Overview of Attack Transcripts*

Now we redefine the transcript random variable of an adversary. We replace ciphertext and tag by all Y-values of encryption queries. Note that ciphertext and tag can be uniquely computed from Y-values. Thus, the transcript of the adversary $\mathcal{T} := (\mathcal{T}_e, \mathcal{T}_f)$ where

1. $\mathcal{T}_e = (\mathsf{N}_i, \mathsf{A}_i, \mathsf{M}_i, \mathsf{Y}_i), i = 1 \ldots q_e$ and
2. $\mathcal{T}_f = (\mathsf{N}^*_{i'}, \mathsf{A}^*_{i'}, \mathsf{C}^*_{i'}, \mathsf{T}^*_{i'}, \mathsf{Z}^*_{i'}), i' = 1 \ldots q_f$.

Here, $\mathsf{Z}^*_{i'}$ denotes the output of the decryption oracle $\mathcal{D}$ (it is always $\perp$ when we interact with the ideal oracle) for the i'th decryption query $(\mathsf{N}^*_{i'}, \mathsf{A}^*_{i'}, \mathsf{C}^*_{i'}, \mathsf{T}^*_{i'})$. Note that Y_i denotes $(\mathsf{Y}_i[0], \ldots, \mathsf{Y}_i[\ell_i]) = \mathsf{Y}_i[0 \ldots \ell_i]$, where $\ell_i = a_i + m_i$, and a_i (resp. m_i) denotes the block length of A_i (resp. M_i). Similarly we define $c^*_{i'}$ and $a^*_{i'}$ for forging queries and write $\ell^*_{i'} = a^*_{i'} + c^*_{i'}$. The values of X_i are uniquely determined from Y_i, A_i, and M_i. We use $\mathcal{T}^{\mathsf{real}} = (\mathcal{T}_e^{\mathsf{real}}, \mathcal{T}_f^{\mathsf{real}})$ (or $\mathcal{T}^{\mathsf{ideal}} = (\mathcal{T}_e^{\mathsf{ideal}}, \mathcal{T}_f^{\mathsf{ideal}})$) to denote the random variable corresponding to the transcripts in the Real world (or in the Ideal world, respectively).

NOTATIONS ON PROBABILITIES REALIZING TRANSCRIPTS Consider a fixed transcript $\mathcal{T} = (\mathcal{T}_e, \mathcal{T}_f)$ where

1. $\mathcal{T}_e = (N_i, A_i, M_i, Y_i)_{i=1 \ldots q_e}$ and
2. $\mathcal{T}_f = (N^*_{i'}, A^*_{i'}, C^*_{i'}, T^*_{i'}, Z^*_{i'})_{i'=1 \ldots q_f}$.

We use the notation $\Pr_{\mathsf{real}}[\mathcal{T}]$ (or $\Pr_{\mathsf{ideal}}[\mathcal{T}]$), to denote the probability

$$\Pr[(\mathsf{N}_i, \mathsf{A}_i, \mathsf{M}_i, \mathsf{Y}_i)_{i=1 \ldots q_e} = \mathcal{T}_e \ \wedge \ (\mathsf{N}^*_{i'}, \mathsf{A}^*_{i'}, \mathsf{C}^*_{i'}, \mathsf{T}^*_{i'}, \mathsf{Z}^*_{i'})_{i'=1 \ldots q_f} = \mathcal{T}_f],$$

in the real world (or in the Ideal world), where the probability is defined over randomness of the transcript. $\mathcal{T}$ is called **realizable** in the real world (or in the Ideal world) if $\Pr_{\mathsf{real}}[\mathcal{T}] > 0$ or $(\Pr_{\mathsf{Ideal}}[\mathcal{T}] > 0$, respectively).

5.6. *Definition and Analysis of Bad Transcripts*

Bad Views A transcript $\mathcal{T} := (\mathcal{T}_e, \mathcal{T}_f)$ is called "bad" if one of the following events occurs:

B1: $\mathsf{L}_i[j] = 0^{n/2}$ for some $i \in [1 \ldots q_e]$ and $j > 0$.

B2: $\mathsf{X}_i[j] = \mathsf{X}_{i'}[j']$ for some $(i, j) \neq (i', j')$ where $j, j' > 0$.[6]
B3: $\mathsf{mcoll}(\mathsf{R}) > n/2$, where R is the tuple of all $\mathsf{R}_i[j]$ values.
B4: $\mathsf{X}^*_i[j] = \mathsf{X}_{i_1}[j_1]$ or $\mathsf{X}^*_i[j] = \mathsf{X}^*_{i'}[j']$ for some i, i', j', i_1, j_1 such that $p_i < j \leq \ell^*_i$ and $\mathsf{prefix}(\mathsf{Y}^*, i, j) \neq \mathsf{prefix}(\mathsf{Y}^*, i', j')$ (where $\mathsf{Y}^*_i[j] = \mathsf{R}(\mathsf{X}^*_i[j])$ and $\mathsf{Y}^*_{i'}[j'] = \mathsf{R}(\mathsf{X}^*_{i'}[j'])$).
B5: There exists i, such that T^*_i is a correct guess of $\mathsf{Y}^*_i[\ell^*_i]$. This clearly cannot happen for the ideal oracle case.
B6: There exists i, such that $\mathsf{Z}^*_i \neq \perp$. This clearly cannot happen for the ideal oracle case.

Some Intuitions on the Bad Events We add some intuitions on these events.

- When **B1** does not hold, then $\mathsf{X}_i[j] \neq \mathsf{X}_{i'}[0]$ for all i, i', and $j > 0$. Hence, Δ_i will be completely random.
- When **B2** does not hold, then all the inputs for the random function are distinct for encryption queries, which makes the responses from encryption oracle completely random in the Real game.
- When **B3** does not hold, then at the right half of $\mathsf{X}_i[j]$ we see at most $n/2$ multi-collisions. A successful forgery is to choose one of the $n/2$ multi-collision blocks and forge the left part so that the entire block collides. Forging the left part has $2^{-n/2}$ probability due to randomness of masking.
- When **B4** does not hold, then the $(p_i + 1)$st to ℓ^*_ith input for the ith forging attempt will be fresh with a high probability and so all the subsequent inputs will remain fresh with a high probability. Also, it ensures that there is no collision between the inputs
- Finally, when **B5** does not hold, then the all the guesses for the tag in the decryption queries are wrong.

If adv interacts with the real oracle, we always have

1. $\mathsf{X}_i[0] = 0^{n/2} \| N_i$
2. $\mathsf{R}(\mathsf{X}_i[j]) = \mathsf{Y}_i[j], i = 1 \ldots q_e, j = 0 \ldots \ell_i$,

where $\mathsf{X}_i[j]$ and $\mathsf{Y}_i[j]$s are computed via ρ and Δ values.

The following lemma bounds the probability of not realizing a good view while interacting with a random function (this will complete the first condition of the Coefficients-H technique).

Lemma 4. *For any transcript $\mathcal{T}$,*

$$\Pr_{\mathsf{ideal}}[\mathcal{T} \notin \mathcal{V}_{good}] \leq \Pr[\boldsymbol{B1}] + \Pr[\boldsymbol{B2}] + \Pr[\boldsymbol{B3}] + \Pr[\boldsymbol{B4} \wedge \boldsymbol{B1}^c \wedge \boldsymbol{B3}^c] + \Pr[\boldsymbol{B5}]$$
$$\leq \frac{4\sigma_e}{2^{n/2}} + \frac{(q_e + \sigma_e + 2\sigma_f) \cdot \sigma_f + q_f}{2^n}.$$

[6] The event **B1** can be captured by **B2** if we allow j' to be zero. However, we do not combine them as they need separate analysis and gives bounds of different orders. Note that **B1** is an event on $n/2$ bits, whereas **B2** is on n bits.

Proof (of Lemma 4) Throughout the proof, we assume all probability notations are defined over the ideal game. We bound all the bad events individually, and then by using the union bound, we will obtain the final bound. We first develop some more notation. Let $(\mathsf{Y}_i^1[j], \mathsf{Y}_i^2[j], \mathsf{Y}_i^3[j], \mathsf{Y}_i^4[j]) \xleftarrow{n/4} \mathsf{Y}_i[j]$. Similarly, we denote $(M_i^1[j], M_i^2[j]) \xleftarrow{n/2} M_i[j]$.

(1) $\Pr[\mathbf{B1}] \leq \sigma/2^{n/2}$: We fix a pair of integers (i, j) for some $i \in [1 \ldots q]$ and $j \in [1 \ldots \ell_i]$. Now, $\mathsf{L}_i[j]$ can be expressed as

$$(\mathsf{Y}_i^2[j-1] \| \mathsf{Y}_i^3[j-1]) \oplus (\alpha^a \cdot (1+\alpha)^b \cdot \Delta_i) \oplus M_i^1[j]$$

for some a and b. Note that when $j > 1$, Δ_i and $\mathsf{Y}_i[j-1]$ are independently and uniformly distributed, and hence for those j, we have $\Pr[\mathsf{L}_i[j] = 0^{n/2}] = 2^{-n/2}$ (apply Lemma 2 after conditioning $\mathsf{Y}_i[j-1]$). Now when $j = 1$, we have the following three possibilities: (i) $\mathsf{L}_i[1] = (1+\alpha) \cdot \Delta_i \oplus \mathsf{Cons}$ if $a_i \geq 2$, (ii) $\mathsf{L}_i[1] = \alpha \cdot \Delta_i \oplus \mathsf{Cons}$ if $a_i = 1$ and the associated data block is full, and (iii) $\mathsf{L}_i[1] = \alpha^2 \cdot \Delta_i \oplus \mathsf{Cons}$ if $a_i = 1$ and the associated data block is not full, for some constant Cons. In all cases by applying Lemma 2, $\Pr[\mathbf{B1}] \leq \sigma_e/2^{n/2}$.

(2) $\Pr[\mathbf{B2}] \leq \sigma_e/2^{n/2}$: For any $(i, j) \neq (i', j')$ with $j, j' \geq 1$, the equality event $\mathsf{X}_i[j] = \mathsf{X}_{i'}[j']$ has a probability at most 2^{-n} since this event is a non-trivial linear equation on $\mathsf{Y}_i[j-1]$ and $\mathsf{Y}_{i'}[j'-1]$ and they are independent to each other. Note that $\sigma_e^2/2^n \leq \sigma_e/2^{n/2}$ as we are estimating probabilities.

(3) $\Pr[\mathbf{B3}] \leq 2\sigma_e/2^{n/2}$: The event $\mathbf{B3}$ is a multi-collision event for randomly chosen σ many $n/2$-bit strings as Y values are mapped in a regular manner (see the feedback function) to R values. From the union bound, we have

$$\Pr[\mathbf{B3}] \leq \binom{\sigma_e}{n/2} \frac{1}{2^{(n/2)\cdot((n/2)-1)}} \leq \frac{\sigma_e^{n/2}}{2^{(n/2)\cdot((n/2)-1)}} \leq \left(\frac{\sigma_e}{2^{(n/2)-1}}\right)^{n/2} \leq \frac{2\sigma_e}{2^{n/2}},$$

where the last inequality follows from the assumption ($\sigma_e \leq 2^{(n/2)-1}$).

(4) $\Pr[\mathbf{B4} \wedge \mathbf{B1}^c \wedge \mathbf{B3}^c] \leq \frac{(q_e+\sigma_e+1.5\sigma_f)\cdot\sigma_f}{2^n}$: We fix some i and want to bound the probability $\Pr[\mathsf{X}_i^*[j] = \mathsf{X}_{i_1}[j_1] \wedge \mathbf{B1}^c \wedge \mathbf{B3}^c]$ for some i_1, j_1 and $p_i < j \leq \ell_i^*$. We also want to bound the probability $\Pr[\mathsf{X}_i^*[j] = \mathsf{X}_{i'}^*[j'] \wedge \mathbf{B1}^c \wedge \mathbf{B3}^c]$ for some i', j' and j. The number of bad pairs is at most

$$(q_e + \sigma_e + \ell_i^*) \cdot \ell_i^* + \sigma_f \cdot \ell_i^* \leq (q_e + \sigma_e + \sigma_f) \cdot \ell_i^* + \sigma_f \cdot \ell_i^*.$$

Therefore, the total number of bad pairs is $\sum_{1 \leq i \leq q_f} (q_e + \sigma_e + \sigma_f) \cdot \ell_i^* + \sigma_f \cdot \ell_i^* \leq (q_e + \sigma_e + 2\sigma_f) \cdot \sigma_f$. Thus, $\Pr[\mathbf{B4} \wedge \mathbf{B1}^c \wedge \mathbf{B3}^c]$ is at most $\frac{(q_e+\sigma_e+2\sigma_f)\cdot\sigma_f}{2^n}$.

(5) $\Pr[\mathbf{B5}] \leq q_f/2^n$: The event $\mathbf{B5}$ is bounded by $\frac{q_f}{2^n}$ since, $Y_i^*[\ell_i^*]$ is uniform, and there are total q_f decryption queries

Summarizing, we have

$$\Pr_{\mathsf{ideal}}[\mathcal{T} \notin \mathcal{V}_{\text{good}}] \le \Pr[\mathbf{B1}] + \Pr[\mathbf{B2}] + \Pr[\mathbf{B3}] + \Pr[\mathbf{B4} \wedge \mathbf{B1}^c \wedge \mathbf{B3}^c] + \Pr[\mathbf{B5}]$$
$$\le \frac{\sigma_e}{2^{n/2}} + \frac{\sigma_e}{2^{n/2}} + \frac{2\sigma_e}{2^{n/2}} + \frac{(q_e + \sigma_e + 2\sigma_f) \cdot \sigma_f}{2^n} + \frac{q_f}{2^n}$$
$$= \frac{4\sigma_e}{2^{n/2}} + \frac{(q_e + \sigma_e + 2\sigma_f) \cdot \sigma_f + q_f}{2^n},$$

which concludes the proof. □

Analysis of Good Transcripts We define a view to be **good** if none of the bad events hold. Let $\mathcal{V}_{\text{good}}$ be the set of all such good views. For any view $\mathcal{T} \in \mathcal{V}_{\text{good}}$, we want to show $\mathsf{ip}_{\mathsf{real}}(\mathcal{T}) \ge (1 - \epsilon_2) \cdot \mathsf{ip}_{\mathsf{ideal}}(\mathcal{T})$ for some ϵ_2. Hence, we proceed by computing the ratio between $\mathsf{ip}_{\mathsf{real}}(\mathcal{T})$ and $\mathsf{ip}_{\mathsf{ideal}}(\mathcal{T})$ for $\mathcal{T} \in \mathcal{V}_{\text{good}}$. Note that, here $\mathsf{ip}_{\mathsf{real}}(\mathcal{T})$ is actually $\Pr_{\mathsf{real}}[\mathcal{T}]$ and $\mathsf{ip}_{\mathsf{ideal}}(\mathcal{T})$ is $\Pr_{\mathsf{ideal}}[\mathcal{T}]$. We first fix a realizable (with respect to ideal world) good view

$$\mathcal{T} = ((N_i, A_i, M_i, Y_i)_{i \in \{1,\ldots,q_e\}}, (N^*_{i'}, A^*_{i'}, C^*_{i'}, T^*_{i'}, Z^*_{i'})_{i' \in \{1,\ldots,q_f\}}),$$

where $Z^*_{i'} = \perp$. We separate $\mathcal{T}$ into

$$\mathcal{T}_e = (N_i, A_i, M_i, Y_i)_{i \in \{1,\ldots,q_e\}} \text{ and } \mathcal{T}_f = (N^*_{i'}, A^*_{i'}, C^*_{i'}, T^*_{i'}, Z^*_{i'})_{i' \in \{1,\ldots,q_f\}},$$

Lemma 5. *For a good and a realizable view $\mathcal{T}$*

$$\Pr_{\mathsf{ideal}}[\mathcal{T}] = 1/2^{n(q_e+\sigma_e)}.$$

Proof of Lemma 5. All the $q_e + \sigma_e$ internal Ys are chosen uniformly from $\{0, 1\}^n$. Hence, it is easy to see that for a good view $\mathcal{T}$, $\Pr_{\mathsf{ideal}}[\mathcal{T}]$ is equal to $1/2^{n(q_e+\sigma_e)}$. □

Lemma 6. *For a good and a realizable view τ,*

$$\Pr_{\mathsf{real}}[\mathcal{T}] = \Pr_{\mathsf{ideal}}[\mathcal{T}].$$

Proof of Lemma 6. Now we consider the real case. Since **B1** and **B2** do not hold with $\mathcal{T}$, all inputs of the random function inside $\mathcal{T}_e$ are distinct, which implies that the released Y-values are independent and uniformly random. The variables in $\mathcal{T}_e$ are uniquely determined given these Y-values, and there are exactly $q_e + \sigma_e$ distinct input–output of R. Therefore, $\Pr_{\mathsf{real}}[\mathcal{T}_e]$ is exactly $2^{-n(q_e+\sigma_e)}$ (note that, $\Pr_{\mathsf{real}}[\mathcal{T}_e]$ is defined exactly in the same way as $\Pr_{\mathsf{real}}[\mathcal{T}]$). We also use the notation $\Pr_{\mathsf{real}}[\mathcal{T}_f|\mathcal{T}_e]$ which is defined in a similar way. We next evaluate

$$\Pr_{\mathsf{real}}[\mathcal{T}] = \Pr_{\mathsf{real}}[\mathcal{T}_e] \cdot \Pr_{\mathsf{real}}[\mathcal{T}_f|\mathcal{T}_e] = \frac{1}{2^{n(q_e+\sigma_e)}} \cdot \Pr_{\mathsf{real}}[\mathcal{T}_f|\mathcal{T}_e]. \tag{6}$$

Lower Bounding $\Pr_{\mathsf{real}}[\mathcal{T}_f|\mathcal{T}_e]$ We observe that $\Pr_{\mathsf{real}}[\mathcal{T}_f|\mathcal{T}_e]$ is equal to $\Pr_{\mathsf{real}}[\bot_{\mathsf{all}}|\mathcal{T}_e]$, where $\bot_{\mathsf{all}}$ denotes the event that $\mathsf{Z}_i^* = \bot$ for all $i = 1, \ldots, q_f$, as other variables in $\mathcal{T}_f$ are determined by $\mathcal{T}_e$.

Let η denote the event that, for all $i = 1, \ldots, q_f$, $\mathsf{X}_i^*[j]$ for $p_i < j \leq \ell_i^*$ is not colliding to X-values in $\mathcal{T}_e$ and $X_i^*[j']$ for all $j' \neq j$. For $j = p_i + 1$, the above condition is fulfilled by **B4**, and thus, $\mathsf{Y}_i^*[p_i + 1]$ is uniformly random, and hence, $\mathsf{X}_i^*[p_i + 2]$ is also uniformly random, due to the property of feedback function (here, observe that the mask addition between the chains of $\mathsf{Y}_i^*[j]$ to $\mathsf{X}_i^*[j+1]$ does not reduce the randomness).

Now we have

$$\Pr_{\mathsf{real}}[\bot_{\mathsf{all}}|\mathcal{T}_e] = 1 - \Pr_{\mathsf{real}}[(\bot_{\mathsf{all}})^c|\mathcal{T}_e],$$

and we also have

$$\Pr_{\mathsf{real}}[(\bot_{\mathsf{all}})^c|\mathcal{T}_e] = \Pr_{\mathsf{real}}[(\bot_{\mathsf{all}})^c, \eta|\mathcal{T}_e] + \Pr_{\mathsf{real}}[(\bot_{\mathsf{all}})^c, \eta^c|\mathcal{T}_e].$$

Here, $\Pr_{\mathsf{real}}[(\bot_{\mathsf{all}})^c, \eta|\mathcal{T}_e]$ is the probability that at least one T_i^* for some $i = 1, \ldots, q_f$ is correct as a guess of $\mathsf{Y}_i^*[\ell_i^*]$. Since **B5** does not hold with τ, the probability $\Pr_{\mathsf{real}}[(\bot_{\mathsf{all}})^c, \eta|\mathcal{T}_e]$ is 0.

For $\Pr_{\mathsf{real}}[(\bot_{\mathsf{all}})^c, \eta^c|\mathcal{T}_e]$ which is at most $\Pr_{\mathsf{real}}[\eta^c|\mathcal{T}_e]$, the above observation suggests that this can be evaluated by counting the number of possible bad pairs (i.e., a pair that a collision inside the pair violates η) among all the X-values in $\mathcal{T}_e$ and all X^*-values in $\mathcal{T}_f$ as well as collisions between two X-values among all the X-values in τ_f as in the same manner to the collision analysis of, e.g., CBC-MAC using R. **B4** does not hold with τ, $\Pr_{\mathsf{real}}[(\bot_{\mathsf{all}})^c, \eta^c|\mathcal{T}_e]$ is 0. Hence, $\Pr_{\mathsf{real}}[\mathcal{T}_f|\mathcal{T}_e]$ is

$$\Pr_{\mathsf{real}}[\mathcal{T}_f|\mathcal{T}_e] = \Pr_{\mathsf{real}}[\bot_{\mathsf{all}}|\mathcal{T}_e] = 1.$$

Combining all, we have

$$\mathsf{ip}_{\mathsf{real}}(\tau) = \Pr_{\mathsf{real}}[\mathcal{T}] = \frac{1}{2^{n(q_e+\sigma_e)}} \cdot \Pr_{\mathsf{real}}[\mathcal{T}_f|\mathcal{T}_e] = \Pr_{\mathsf{ideal}}[\tau] = \mathsf{ip}_{\mathsf{ideal}}(\tau).$$

□

6. Hardware Implementation of COFB

6.1. *Overview*

COFB primarily aims to achieve a lightweight implementation on small hardware devices. For such devices, the hardware resource for implementing memory is often the dominant factor of the size of entire implementation, and the scalability by paral-

Table 2. Clock cycles per message byte for COFB[AES].

	Message length (Bytes)										
	16	32	64	128	256	512	1024	2048	4096	16384	32768
cpb	2.93	2.22	1.86	1.68	1.59	1.54	1.52	1.51	1.50	1.50	1.50

Table 3. Clock cycles per message byte for COFB[GIFT].

	Message length (Bytes)										
	16	32	64	128	256	512	1024	2048	4096	16384	32768
cpb	5.441	5.283	5.204	5.164	5.145	5.135	5.130	5.127	5.126	5.125	5.125

lelizing the internal components is not needed. In this respect, COFB's small-state size and completely serial operation are quite desirable.

For implementation aspects, COFB is simple, as it consists of a blockcipher and several basic operations (bitwise XOR, the feedback function, and the constant multiplications over $\mathrm{GF}(2^{n/2})$). Combined with the small-state size, this implies that the implementation size of COFB is largely dominated by the underlying blockcipher. In this section, we provide hardware implementation details of COFB using two blockciphers, AES and GIFT. Here, GIFT is a family of lightweight blockcipher proposed by Banik et al. [15]. It employs a structure similar to PRESENT [21] while improves efficiency by carefully choosing S-box and the bit permutation. It has 64-bit and 128-bit block versions, both have 128-bit key. We write GIFT-128 or simply write GIFT to denote the 128-bit block version. We write COFB[AES] and GIFT-128 to denote COFB using AES-128 and COFB[GIFT], respectively.

We provide the number of clock cycles needed to process input bytes, as a conventional way to estimate the speed. Here, COFB[AES] taking a-block AD (associated data) and an m-block message needs $12(a+m)+23$ cycles. Table 2 shows the number of average cycles per input message bytes, which we call cycles per byte (cpb), assuming AD has the same length as message and the underlying blockcipher has 128-bit block. That is, the table shows $(12 \cdot 2m + 23)/16m$.

Similarly, COFB[GIFT] needs $41 \cdot (a+m)+81$ cycles for a-block AD and an m-block message. Table 3 shows the number of average cycles per input message bytes, which we call cycles per byte (cpb), assuming AD has the same length as message and the underlying blockcipher has 128-bit block. That is, the table shows $(41 \cdot 2m + 81)/16m$.

6.2. *Hardware Implementation of COFB[BC] Without CAESAR Hardware API*

We describe the implementation details of both COFB[AES] and COFB[GIFT] without the CAESAR hardware API. These are basic round-based implementations without any pipelining and employ module architecture. We primary focus on the encryption-only circuit; however, the combined encryption and decryption circuit should have very small amount of overhead thanks to the inverse freeness (i.e., no blockcipher decryption routine

is needed) and simplicity of the mode. Due to the similarity between the associated data and the message processing phase, the same hardware modules are used in both phases. A single bit switch is used to distinguish between the two types of input data. The main architecture consists of the modules described below. We remark that there is also a Finite State Machine (FSM) which controls the flow by sending signal to these modules. The FSM has a rather simple structure and is described below. Then, the overall hardware architecture is described in Fig. 8. We would like to mention that both the versions can be described with the same hardware architecture as they have exactly the same interface. Hence, we often use BC instead of the underlying blockcipher, where $BC \in$ {AES-128, GIFT-128}. We also assume that BC comprises of r rounds.

1. **State Registers** The state registers are used to store the intermediate states after each iteration. We use a 128-bit **State** register to store the 128-bit BC block state, a 64-bit Δ register to store the 64-bit mask applied to each BC input, and a 128-bit **Key** register to store the 128-bit key. The round key of BC is stored in the additional 128-bit register (**Round Key**); however, this is included in the BC module.
2. **BC Round** BC round function module runs one BC round computation and produces a 128-bit output, using two 128-bit inputs: one from the **State** and the other from (internal) **Round Key** registers. The latter register is initialized by loading the master key, stored in the **Key** register, each time the BC function is invoked. The output of BC module is stored into the **State** register, which is the input for the next round. The entire operation is serial, while the internal round computation and the round key generation run in parallel, and needs $r + 1$ cycles to perform full BC encryption.
3. **Feedback Function** ρ. The ρ module is to compute the linear feedback function ρ on the 128-bit data block and the 128-bit intermediate state value (output from the BC computation). The output is a 128-bit ciphertext and a 128-bit intermediate state (to be masked and stored to the **State** register).
4. **Mask Update uMask** module updates the mask stored in Δ register. **uMask** receives the current mask value and updates it by multiplying with α or $(1+\alpha)$ or $(1+\alpha)^2$ based on the signals generated by the FSM, where signals are to indicate the end of the message and the completeness of the final block process.
5. **FSM** The control of the complete design can be described by a finite state machine (FSM). We provide a separate and simple view of FSM in Fig. 7. The FSM consists of 9 states and starts with the $Reset_St$. This state is idle and followed by a $Load_St$, which initializes the BC state by loading nonce (before the first BC invocation). After the initialization, FSM enters into the BC invocation phase to encrypt the nonce. This phase consists of BC_Reset_St to reset BC parameters, BC_Start_St for key whitening, BC_Round_St to run one BC round and BC_Done_St to indicate the end of the BC invocation. Depending on whether the current blockcipher call is final or not, the FSM either releases the tag or it enters to the $Compute_\rho_Add_Mask_St$, which computes the ρ function, updates mask, and partially masks the blockcipher input. The FSM sends two additional bits EOM to denote the end of data block and isComplete to denote the last data block is complete or not. Next it enters the BC_Reset_St for the next blockcipher invocation. After the last BC invocation, it enters the $Release_Tag_St$. Finally, the

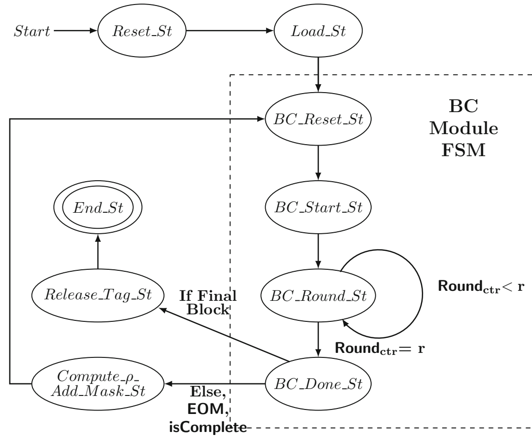

Fig. 7. FSM for COFB[BC] hardware implementation.

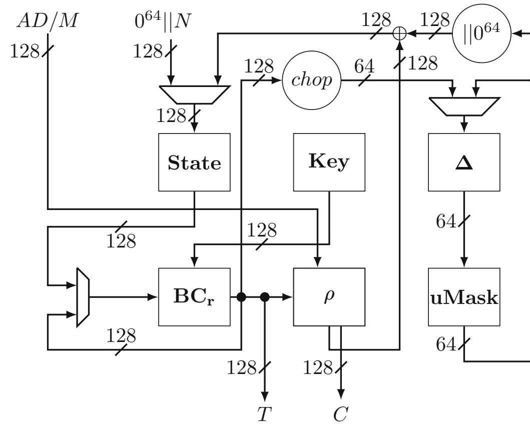

Fig. 8. Hardware circuit diagram.

FSM enters the end state. We use a 4-bit register to keep track of the states. It is to be noted that, in addition to the state transition, FSM also sends the corresponding relevant signals to the top modules.

Basic Implementation We describe a basic flow of our implementation, which generally follows the pseudocode of Fig. 5. Prior to the initialization, **State** register is loaded with $0^{64} \parallel N$. Once **State** register is initialized, the initialization process starts by encrypting the nonce ($0^{64} \parallel N$) with BC. Then, 64 bits of the encrypted nonce is chopped by the "chop" function as shown in Fig. 8, and this chopped value is stored into the Δ register (this is initialization of Δ). After the initialization, 128-bit associated data blocks are fetched and sent to the ρ module along with the previous BC output to produce a 128 bit intermediate state. This state is partially masked with 64-bit Δ for every BC call. After all the associated data blocks are processed, the message blocks are processed in the

Table 4. FPGA implementation results of COFB[AES] and COFB[GIFT].

Design (platform)	Slice registers	LUTs	Slices	Frequency (MHz)	Throughput (Gbps)	Mbps/LUT	Mbps/slice
COFB[AES] (Virtex 6)	594	1051	449	267.20	2.85	2.71	6.35
COFB[AES] (Virtex 7)	593	1440	564	274.84	2.93	2.03	5.19
COFB[GIFT] (Virtex 6)	342	771	355	612.91	1.91	2.48	5.51
COFB[GIFT] (Virtex 7)	342	771	316	712.99	2.23	2.89	6.62

same manner, except that the ρ function produces 128-bit ciphertext blocks in addition to the intermediate state values. Finally, after the message processing is over, the tag is generated using an additional BC call.

Combined Encryption and Decryption As mentioned earlier, we here focus on the encryption-only circuit. However, due to the similarity between the encryption and the decryption modes, the combined hardware for encryption and decryption can be built with a small increase in the area, with the same throughput. This can be done by adding a control flow to a binary signal for mode selection.

6.3. *Implementation Results with out CAESAR Hardware API*

We have implemented both COFB[AES] and COFB[GIFT] on Xilinx Virtex 6 and Virtex 7, using VHDL and Xilinx ISE 13.4. Table 4 presents the implementation results of COFB on Virtex 7 with the target device xc7vx330t and on Virtex 6 with the target device xc6vlx760. We employ RTL approach and a basic iterative type architecture (128-bit round- based implementation). The areas are listed in the number of Slice Registers, Slice LUTs and Occupied Slices. We also report frequency (MHz), Throughput (Gbps), and throughput–area efficiency. Table 4 presents the mapped hardware results of COFB[AES]. In this paper, we have slightly optimized the implementation in [25,26] to get a better estimate of the number of slice registers.

For AES-128, we use the implementation available from Athena [1] maintained by George Mason University. This implementation stores all the round subkeys in a single register to make the AES implementation faster and parallelizable. However, the main motivation of COFB is to reduce hardware footprint. Hence, we change the above implementation to a sequential one such that it processes only one AES round in a single clock cycle. This in turn eliminates the need to store all the round subkeys in a single register and reduces the hardware area consumed by the AES module.

For GIFT-128, we use our own implementation in FPGA. The implementation is round based without any pipelining. The architecture uses three registers $State$, RK, and $Round$ to hold the blockcipher state, current round key, and the round counter, respectively. The architecture is divided into four modules SN, BP, ARK and ARC, $UKEY$. operations. SN module applies a 4-bit s-box to each of the 4-bit nibbles of the

Table 5. COFB[AES] and COFB[GIFT]: area utilization by modules in Virtex 6.

Modules	Slices	LUTs
COFB[AES]		
Total	449	1051
AES_r	311	657
Others	138	394
COFB[GIFT]		
Total	355	771
GIFT_r	155	346
Others	200	425

state. BP applies the bit permutation on the state. ARK performs the round key addition on the state, and ARC applies round constant addition on the state. $UKEY$ updates the round key and stores it in RK. The architecture also uses another module EXT to extract a part of the round key to be added to the state. The hardware implementation results in slice registers, slice LUTs, and Slices are presented in Table 4.

6.4. *Hardware Flexibility of the COFB Design*

COFB is itself very lightweight, and it uses a few operations other than the blockcipher computations. In Table 5, we present the hardware area occupied by the blockcipher and the other modules for both COFB[AES] and COFB[GIFT] on Vertex 6. We observe that COFB[AES] consumes low hardware footprint and the majority of the hardware footprint is used by AES, whereas in COFB[GIFT] the implementation size is much smaller as the underlying blockcipher GIFT is much lighter than AES. This depicts that implementation area optimized blockcipher will be the most efficient one.

6.5. *Hardware Implementation of COFB[BC] with CAESAR Hardware API*

We implement COFB[BC] (both encryption and decryption) using the CAESAR Hardware API with the motivation of a fair comparison. We would like to mention that the architecture described in Sect. 6.2 is different from this one in terms of the user interface and the design of the control unit. For example, in the previous implementation, we assume that the nonce, the associated data blocks, and the message blocks are supplied separately to the main module. In this case, all of them are supplied as input blocks with a corresponding data type. Also, the previous architecture is encryption only, whereas this architecture supports both encryption and decryption. The implementation with GMU API adds a significant overhead on the previous result.

The top-level module is AEAD, which invokes 4 low-level modules PreProcessor, PostProcessor, CMD FIFO, and Cipher-Core. We primarily focus on the Cipher-core module, as the GMU package [7] already provides codes for the other modules. We do not need to concentrate on the internal structures of all the modules except the Cipher-Core one. However, we just need to set certain parameters according to our design criteria. A top-level block diagram of this API is given in [41].

Cipher-core Module The Cipher-core module describes the core architecture of COFB[BC]. It invokes Cipher-core-controller and Cipher-core-datapath modules. The controller describes the finite state machine for the design, and the datapath describes the data processing flow between the internal circuit and sub-modules. The Cipher-core-controller and the Cipher-core-datapath modules are described below.

Cipher-core-controller Module The Cipher-core-controller module describes the finite state machine (FSM) for the CAESAR hardware-based implementation of COFB. At the top layer, the controller uses the standard state descriptions provided in specification for the CAESAR hardware API [41] along with a few extra state compatible to our design. The standard states are as mentioned below

- S_RESET
- S_KEY_CHECK
- S_NPUB_READ
- S_DATA_LOAD
- $S_AD_PROCESS$
- $S_AD_PROCESS_LAST_BLOCK$
- $S_DATA_PROCESS$
- $S_DATA_PROCESS_LAST_BLOCK$
- $S_GEN_VER_TAG$.

Our design requires a few extra states

- $S_NONCE_PROCESS$
- S_AES_START
- S_AES_DONE
- $S_COMPUTE_RHO_AND_TWEAK$.

The internal BC module runs its own FSM with BC_Reset_St, BC_Start_St, BC_Round_St and BC_Done_St. The FSM for the BC module follows the same described in Sect. 6.2.

The control enters through the S_RESET state and consequently passes through S_KEY_CHECK and S_NPUB_READ states according to the specification. S_NPUB_READ is followed by S_DATA_LOAD which in turn is followed by $S_NONCE_PROCESS$, $S_AD_PROCESS$, or $S_DATA_PROCESS$ depending on the data type. If the block data input is not valid, the control returns to S_DATA_LOAD. Otherwise, from each of these 3 states, the control enters S_BC_START to denote a BC invocation. Note that, there are four intermediate states between S_BC_START and S_BC_DONE corresponding to the BC module. After BC computation is complete (as signaled by S_BC_DONE), the control goes to $S_COMPUTE_RHO_AND_TWEAK$. For the last input block of a certain data type, the control enters $S_AD_PROCESS_LAST_BLOCK$ (or $S_DATA_PROCESS_LAST_BLOCK$). If it is not the last block, the control reverts back to $S_AD_PROCESS$ (or $S_DATA_PROCESS$, respectively). Finally, after the message is processed the FSM enters $S_GEN_VER_TAG$, where the tag is either generated (for encryption) or verified (for decryption).

Table 6. Implementation results of COFB[AES] and COFB[GIFT] using CAESAR hardware API.

Platform	Design	Slice registers	LUTs	Slices	Frequency (MHz)	Throughput (Gbps)	Mbps/ LUT	Mbps/ slice
Virtex 6	COFB[AES]-CAESAR-API	1210	1475	584	251.848	2.686	1.823	4.360
Virtex 7	COFB[AES]-CAESAR-API	1209	1496	579	257.486	2.747	1.842	4.395
Spartan 6	COFB[AES]-CAESAR-API	1219	1544	590	141.448	1.509	0.977	2.558
Virtex 6	COFB[GIFT]-CAESAR-API	1152	1011	415	352.392	1.100	1.045	2.723
Virtex 7	COFB[GIFT]-CAESAR-API	1150	1041	355	373.002	1.164	1.172	2.604
Spartan 6	COFB[GIFT]-CAESAR-API	1156	1040	385	149.477	0.467	0.449	1.213

We denote the implementation of COFB[AES] with CAESAR API by COFB[AES]-CAESAR-API (same for COFB[GIFT])

Control Signals The controller uses signals described in [41] as well as a few additional signals. The controller accepts the following additional input signals:

1. bc_done (to denote the completion of BC invocation) and
2. tag_match (to denote verification).

It outputs the following additional output signals:

1. delta_load signal (to denote whether to load the Δ register or to update it)
2. sel_ed signal (to denote whether the input to the BC module is the nonce or the internal feedback value)
3. bc_start (to denote the invocation of the blockcipher)
4. init_key (to initialize the key register)

Cipher-core-datapath Module. This module accepts inputs from both the Cipher-core and the Cipher-core-controller modules. It accepts the secret key, nonce, message blocks, associate data blocks, and several other control signals described in the specification [41]. It also accepts additional signals delta_load, sel_ed, bc_start, and init_key generated by Cipher-core-controller. It outputs the ciphertext block, the tag, and two additional signals bc_done and tag_match. It invokes internal modules for BC round functions, ρ computations, and Δ update functions. These modules are already described in Sect. 6.2.

6.6. *Implementation Results with CAESAR Hardware API*

We implement both COFB[AES] and COFB[GIFT] using CAESAR hardware API on the same platform. We also implement our design on Spartan 6 with the target device xc6slx150t for the sake on comparison with the other results on Spartan 6. Table 6 presents the detailed implementation results for COFB[AES] and COFB[GIFT].

Table 7. Comparison on Virtex 6 [2] (Results are taken by selecting **CAESAR Round 2** and **Standards** implemented results (following CAESAR API) in [2]) .

Scheme	Primitive	LUT	Slices	T'put (Gbps)	Mbps/LUT	Mbps/slice
ACORN [65]	SC	455	135	3.112	6.840	23.052
AEGIS [67]	BC-RF	7592	2028	70.927	9.342	34.974
AES-COPA [11]	BC	7754	2358	2.500	0.322	1.060
AES-GCM [31]	BC	3175	1053	3.239	1.020	3.076
AES-OTR [50]	BC	5102	1385	2.741	0.537	1.979
AEZ [37]	BC-RF	4597	1246	8.585	0.747	2.756
ASCON-128 [30]	Sponge	1271	413	3.172	2.496	7.680
ASCON-128a [30]	Sponge	1587	547	5.099	3.213	9.322
CLOC-AES [40]	BC	3145	891	2.996	0.488	1.724
CLOC-TWINE [40]	BC (non-AES)	1689	532	0.343	0.203	0.645
CLOC-AES-Optimized [40,45]	BC	–	595	0.695	–	1.17
DEOXYS [42]	TBC	3143	951	2.793	0.889	2.937
ELmD [28]	BC	4302	1584	3.168	0.736	2.091
JAMBU-AES [66]	BC	1836	652	1.999	1.089	3.067
JAMBU-SIMON [66]	BC (non-AES)	1222	453	0.363	0.297	0.801
Joltik [41]	TBC	1292	442	0.853	0.660	0.826
Ketje-Jr [20]	Sponge	1236	412	2.832	2.292	6.875
Ketje-Sr [20]	Sponge	1903	613	5.772	3.033	9.416
Minalpher [60]	BC (non-AES)	2879	1104	1.831	0.636	1.659
NORX [12]	Sponge	2964	1016	11.029	3.721	10.855
NORX-Optimized [12,45]	Sponge	–	2398	40.960	–	17.09
PRIMATES-HANUMAN [9]	Sponge	1012	390	0.964	0.953	2.472
OCB [44]	BC	4249	1348	3.122	0.735	2.316
SCREAM [35]	TBC	2052	834	1.039	0.506	1.246
SILC-AES [40]	BC	3066	921	4.040	1.318	4.387
SILC-LED [40]	BC (non-AES)	1685	579	0.245	0.145	0.422
SILC-PRESENT [40]	BC (non-AES)	1514	548	0.407	0.269	0.743
Tiaoxin [52]	BC-RF	7123	2101	52.838	7.418	25.149
TriviA-ck [27]	SC	2118	687	15.374	7.259	22.378
COFB[AES]	BC	1051	449	2.850	2.710	6.350
COFB[AES]-CAESAR-API	BC	1475	584	2.686	1.823	4.360
COFB[GIFT]	BC	771	355	1.911	2.484	5.511
COFB[GIFT]-CAESAR-API	BC	1011	415	1.100	1.045	2.723

In the "Primitive" column, SC denotes stream cipher, (T)BC denotes (Tweakable) blockcipher, and BC-RF denotes the blockcipher's round function. The results for CLOC-AES-Optimized and NORX-Optimized implementations (based on custom API, but not CAESAR API) have been taken from [45]. '–' implies data not available

6.7. *Area Overhead Due to CAESAR Hardware API*

We observe from Tables 4 and 6 that there are significant overheads both in hardware area (for example, 40% increase in LUTs , 30% increase in Slices for COFB[AES] in Virtex 6) and in throughput (for example, 42% decrease in throughput for COFB[GIFT] in Virtex 6). Though CAESAR API favors basic iterative architecture, strict requirements of the complex control unit design and input handling mechanism cuts the efficiency significantly.

Table 8. Comparison on Virtex 7 [2].

Scheme	LUT	Slices	T'put (Gbps)	Mbps/LUT	Mbps/slice
ACORN	499	155	3.437	6.888	22.174
AEGIS	7504	1983	94.208	12.554	47.508
AES-COPA	7795	2221	2.770	0.355	1.247
AES-GCM	3478	949	3.837	1.103	4.043
AES-OTR	4263	1204	3.187	0.748	2.647
AEZ	4686	1645	8.421	0.719	2.047
ASCON-128	1373	401	3.852	2.806	9.606
ASCON-128a	1836	506	5.476	2.982	10.821
CLOC-AES	3552	1087	3.252	0.478	1.561
CLOC-TWINE	1552	439	0.432	0.278	0.984
DEOXYS	3234	954	1.472	0.455	2.981
ELmD	4490	1306	4.025	0.896	3.082
JAMBU-AES	1595	457	1.824	1.144	3.991
JAMBU-SIMON	1200	419	0.368	0.307	0.878
Joltik	1261	390	0.402	0.319	1.031
Ketje-Jr	1567	518	4.080	2.604	7.876
Ketje-Sr	2592	724	6.752	2.605	9.326
Minalpher	2941	802	2.447	0.832	3.051
NORX	2881	857	10.328	3.585	12.051
PRIMATES-HANUMAN	1148	370	1.072	0.934	2.897
OCB	4269	1228	3.608	0.845	2.889
SCREAM	2315	696	1.100	0.475	1.580
SILC-AES	3040	910	4.365	1.436	4.796
SILC-LED	1682	524	0.267	0.159	0.510
SILC-PRESENT	1514	484	0.479	0.316	0.990
Tiaoxin	7556	1985	75.776	10.029	38.174
TriviA-ck	2221	684	14.852	6.687	21.713
COFB[AES]	1440	564	2.933	2.031	5.191
COFB[AES]-CAESAR-API	1496	579	2.747	1.842	4.395
COFB[GIFT]	771	316	2.230	2.892	6.623
COFB[GIFT]-CAESAR-API	1041	355	1.164	1.174	2.604

6.8. *Benchmarking with ATHENa Database*

We compare implemented results of COFB[BC] with the results published in ATHENa Database [2], taking Virtex 6 and Virtex 7 as our target platforms. We also include the optimized results from the paper [45] for benchmarking only in Virtex 6 platform. In the first implementation, we ignore the overhead to support the CAESAR API and the fact that ours is encryption only while the others are (to the best of our knowledge) supporting both encryption and decryption, and the difference in the achieved security level, both quantitative and qualitative. To provide a fair benchmarking, we also provide a CAESAR hardware API-based implementations of COFB[BC]. We found that, even if the implementation results gain a significant overhead, still we achieve a highly competitive result, even after adding a circuit for supporting CAESAR API and decryption. In Table 7, we provide comparisons for Vertex 6, and in Table 8, we provide comparisons for Vertex 7. Note that, we also add two custom API-based implementation results (for CLOC-AES-Optimized and NORX-Optimized) provided in [45]. The other results

Table 9. Comparison on Spartan 6 using results from [2,32,68] .

Scheme	API	LUT	Slices	T'put (Gbps)	Mbps/LUT	Mbps/slice
ACORN	Lightweight API	418	133	1.226	2.932	9.215
NORX	Lightweight API	1424	391	2.989	2.099	7.645
CLOC-AES	Lightweight API	1604	554	0.069	0.043	0.124
SILC-AES	Lightweight API	1052	335	0.077	0.073	0.229
SILC-LED	Lightweight API	872	235	0.015	0.017	0.064
ASCON-128	Lightweight API	684	231	0.060	0.088	0.26
ASCON-128a	Lightweight API	684	231	0.119	0.174	0.52
Ketje-Sr	Lightweight API	450	155	0.024	0.053	0.16
ACORN	CAESAR API	1024	–	3.986	3.826	–
NORX	CAESAR API	3065	–	5.125	1.672	–
CLOC-AES	CAESAR API	3147	–	0.709	0.208	–
SILC-AES	CAESAR API	3404	–	0.707	0.225	–
SILC-LED	CAESAR API	1575	–	0.106	0.067	–
ASCON-128	CAESAR API	1401	–	1.906	1.360	–
ASCON-128a	CAESAR API	1712	–	2.884	1.684	–
Ketje-Sr	CAESAR API	2415	–	3.979	1.648	–
JAMBU-SIMON	CAESAR API	1376	456	0.186	0.135	0.408
COFB[AES]-CAESAR-API	CAESAR API	1544	590	1.509	0.977	2.558
COFB[GIFT]-CAESAR-API	CAESAR API	1040	385	0.467	0.449	1.213

'–' implies data not available

in [45] have been collected in ASIC platform and hence are not compatible with our figures.

Finally, in Table 9, we provide a benchmark of COFB[BC]-CAESAR-API on Spartan 6. We use the results in [32,68] as well as few others (such as JAMBU-SIMON) in [2]. We observe that, some of the implementations (like ASCON, SILC, Ketje) in this platform support a specialized lightweight API (also support CAESAR API) and hence achieve better hardware area than ours (supports standard CAESAR hardware API) following basic iterative architecture.

We also remark that it is basically hard to compare COFB using AES-128 or GIFT-128 with other non-block-cipher-based AE schemes in the right way, because of the difference in the primitives and the types of security guarantee. For example, ACORN is built from scratch and does not have any provable security result and is subjected to several cryptanalyses [29,46,58,59]. Sponge AE schemes (ASCON, Ketje, NORX, and PRIMATES-HANUMAN) use a keyless permutation of a large block size to avoid key scheduling circuit and have the provable security relying on the random permutation model.

7. Conclusion

This paper presents COFB, a blockcipher mode for AE focusing on minimizing the state size. When instantiated with an n-bit blockcipher, COFB operates at rate-1 and requires state size of $1.5n$ bits and is provable secure up to $O(2^{n/2}/n)$ queries based on the standard PRP assumption on the blockcipher. In fact this is the first scheme

fulfilling these features at once. A key idea of COFB is a new type of feedback function combining both plaintext and ciphertext blocks. We first present an idealized version of COFB, named iCOFB along with its provable security analysis. We instantiate COFB with the AES-128 blockcipher. We also present hardware implementation results for COFB with AES-128 and GIFT-128 blockcipher, respectively (both with or without CAESAR Hardware API). These two implementations demonstrate the effectiveness of our approach.

References

[1] ATHENa: Automated Tool for Hardware Evaluation. https://cryptography.gmu.edu/athena/.

[2] Authenticated Encryption FPGA Ranking. https://cryptography.gmu.edu/athenadb/fpga_auth_cipher/rankings_view.

[3] CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. http://competitions.cr.yp.to/caesar.html/.

[4] Recommendation for Block Cipher Modes of Operation: Methods and Techniques. NIST Special Publication 800-38A, 2001. National Institute of Standards and Technology.

[5] Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality . NIST Special Publication 800-38C, 2004. National Institute of Standards and Technology.

[6] Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication. NIST Special Publication 800-38B, 2005. National Institute of Standards and Technology.

[7] CAESAR Development Package. 2016. https://cryptography.gmu.edu/athena/index.php?id=download.

[8] NIST FIPS 197. Advanced Encryption Standard (AES). *Federal Information Processing Standards Publication*, 197, 2001.

[9] Elena Andreeva, Begül Bilgin, Andrey Bogdanov, Atul Luykx, Florian Mendel, Bart Mennink, Nicky Mouha, Qingju Wang, and Kan Yasuda. PRIMATEs v1.02. Submission to CAESAR. 2016. https://competitions.cr.yp.to/round2/primatesv102.pdf.

[10] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Elmar Tischhauser, and Kan Yasuda. Parallelizable and authenticated online ciphers. In *ASIACRYPT (1)*, volume 8269 of *LNCS*, pages 424–443. Springer, 2013.

[11] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Elmar Tischhauser, and Kan Yasuda. AES-COPA v.2. Submission to CAESAR, 2015. https://competitions.cr.yp.to/round2/aescopav2.pdf.

[12] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. NORX v3.0. Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/norxv30.pdf.

[13] Subhadeep Banik, Andrey Bogdanov, and Kazuhiko Minematsu. Low-Area Hardware Implementations of CLOC, SILC and AES-OTR. DIAC, 2015.

[14] Subhadeep Banik, Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, Mridul Nandi, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT-COFB v1.0. https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/GIFT-COFB-spec.pdf.

[15] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A small present—towards reaching the limit of lightweight encryption. In Fischer and Homma [33], pages 321–345.

[16] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Siang Meng Sim, Yosuke Todo, and Yu Sasaki. GIFT: A small present. *IACR Cryptol ePrint Arch.*, 2017:622, 2017.

[17] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK lightweight block ciphers. In *Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, June 7–11, 2015*, pages 175:1–175:6. ACM, 2015.

[18] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology—CRYPTO 2016—-36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14–18, 2016,*

Proceedings, Part II, volume 9815 of *Lecture Notes in Computer Science*, pages 123–153. Springer, 2016.

[19] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of the cipher block chaining message authentication code. *J. Comput. Syst. Sci.*, 61(3):362–399, 2000.

[20] Guido Bertoni, Michaël Peeters Joan Daemen, Gilles Van Assche, and Ronny Van Keer. Ketje v2. Submission to CAESAR. 2016. https://competitions.cr.yp.to/round3/ketjev2.pdf.

[21] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: an ultra-lightweight block cipher. In *CHES 2007*, pages 450–466, 2007.

[22] Andrey Bogdanov, Florian Mendel, Francesco Regazzoni, Vincent Rijmen, and Elmar Tischhauser. ALE: AES-based lightweight authenticated encryption. In *FSE 2013*, pages 447–466, 2013.

[23] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE—a low-latency block cipher for pervasive computing applications—extended abstract. In *ASIACRYPT 2012*, pages 208–225, 2012.

[24] Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN—a family of small and efficient hardware-oriented block ciphers. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6–9, 2009, Proceedings*, volume 5747 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2009.

[25] Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, and Mridul Nandi. Blockcipher-based authenticated encryption: how small can we go? In Fischer and Homma [33], pages 277–298.

[26] Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, and Mridul Nandi. Blockcipher-based authenticated encryption: how small can we go? *IACR Cryptol. ePrint Arch.*, 2017:649, 2017.

[27] Avik Chakraborti and Mridul Nandi. TriviA-ck-v2. Submission to CAESAR. 2015. https://competitions.cr.yp.to/round2/triviackv2.pdf.

[28] Nilanjan Datta and Mridul Nandi. Proposal of ELmD v2.1. Submission to CAESAR, 2015. https://competitions.cr.yp.to/round2/elmdv21.pdf.

[29] Prakash Dey, Raghvendra Singh Rohit, and Avishek Adhikari. (2016) Full key recovery of ACORN with a single fault. *J. Inf. Sec. Appl.*, 29,57–64

[30] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2. Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/asconv12.pdf.

[31] Morris Dworkin. Recommendation for block cipher modes of operation: Galois/counter mode (GCM) and GMAC. NIST Special Publication 800-38D, 2011. csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf.

[32] Farnoud Farahmand, William Diehl, Abubakr Abdulgadir, Jens-Peter Kaps, and Kris Gaj. Improved lightweight implementations of CAESAR authenticated ciphers. *IACR Cryptol. ePrint Arch.*, 2018:573, 2018.

[33] Wieland Fischer and Naofumi Homma, editors. *Cryptographic Hardware and Embedded Systems—CHES 2017—19th International Conference, Taipei, Taiwan, September 25–28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*. Springer, 2017.

[34] Ewan Fleischmann, Christian Forler, and Stefan Lucks. McOE: a family of almost foolproof on-line authenticated encryption schemes. In *FSE 2012*, pages 196–215, 2012.

[35] Vincent Grosso, Gaëtan Leurent, Francois-Xavier Standaert, Kerem Varici, Anthony Journault, Francois Durvaux, Lubos Gaspar, and Stéphanie Kerckhof. SCREAM Side-Channel Resistant Authenticated Encryption with Masking. Submission to CAESAR, 2015. https://competitions.cr.yp.to/round2/screamv3.pdf.

[36] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED block cipher. In *CHES 2011*, pages 326–341, 2011.

[37] Viet Tung Hoang, Ted Krovetz, and Philip Rogaway. AEZ v4.2: Authenticated Encryption by Enciphering. Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/aezv42.pdf.

[38] Tetsu Iwata and Kaoru Kurosawa. OMAC: One-key CBC MAC. In *FSE*, pages 129–153, 2003.

[39] Tetsu Iwata, Kazuhiko Minematsu, Jian Guo, and Sumio Morioka. CLOC: authenticated encryption for short input. In *FSE 2014*, pages 149–167, 2014.

[40] Tetsu Iwata, Kazuhiko Minematsu, Jian Guo, Sumio Morioka, and Eita Kobayashi. CLOC and SILC. Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/clocsilcv3.pdf.

[41] Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. Joltik v1.3. Submission to CAESAR, 2015. https://competitions.cr.yp.to/round2/joltikv13.pdf.

[42] Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. Deoxys v1.41. Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/deoxysv141.pdf.

[43] Ted Krovetz and Phillip Rogaway. The software performance of authenticated-encryption modes. In *FSE*, pages 306–327, 2011.

[44] Ted Krovetz and Phillip Rogaway. OCB(v1.1). Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/ocbv11.pdf.

[45] Sachin Kumar, Jawad Haj-Yihia, Mustafa Khairallah, and Anupam Chattopadhyay. A comprehensive performance analysis of hardware implementations of CAESAR candidates. *IACR Cryptol. ePrint Arch.*, 2017:1261, 2017.

[46] Frédéric Lafitte, Liran Lerman, Olivier Markowitch, and Dirk Van Heule. SAT-based cryptanalysis of ACORN. *IACR Cryptol. ePrint Arch.*, 2016:521, 2016.

[47] Moses Liskov, Ronald L. Rivest, and David A. Wagner. Tweakable block ciphers. In Moti Yung, editor, *Advances in Cryptology—CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2002.

[48] Kerry A. McKay, Larry Bassham, Meltem Snmez Turan, and Nicky Mouha. Report on Lightweight Cryptography, 2017. http://nvlpubs.nist.gov/nistpubs/ir/2017/NIST.IR.8114.pdf.

[49] Kazuhiko Minematsu. Parallelizable rate-1 authenticated encryption from pseudorandom functions. In *EUROCRYPT*, volume 8441 of *LNCS*, pages 275–292. Springer, 2014.

[50] Kazuhiko Minematsu. AES-OTR v3.1. Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/aesotrv31.pdf.

[51] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the limits: a very compact and a threshold implementation of AES. In *EUROCRYPT 2011*, pages 69–88, 2011.

[52] Ivica Nikolić. Tiaoxin – 346. Submission to CAESAR. 2016. https://competitions.cr.yp.to/round3/tiaoxinv21.pdf.

[53] J. Patarin. Etude des Générateurs de Permutations Basés sur le Schéma du D.E.S. Ph.d. Thèsis de Doctorat de l'Université de Paris 6, 1991.

[54] Thomas Peyrin, Siang Meng Sim, Lei Wang, and Guoyan Zhang. Cryptanalysis of JAMBU. In *FSE 2015*, pages 264–281, 2015.

[55] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In Pil Joong Lee, editor, *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5–9, 2004, Proceedings*, volume 3329 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2004.

[56] 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, 2003.

[57] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In *EUROCRYPT*, pages 373–390, 2006.

[58] Md. Iftekhar Salam, Harry Bartlett, Ed Dawson, Josef Pieprzyk, Leonie Simpson, and Kenneth Koon-Ho Wong. Investigating cube attacks on the authenticated encryption stream cipher ACORN. In *ATIS 2016*, pages 15–26, 2016.

[59] Md. Iftekhar Salam, Kenneth Koon-Ho Wong, Harry Bartlett, Leonie Ruth Simpson, Ed Dawson, and Josef Pieprzyk. Finding state collisions in the authenticated encryption stream cipher ACORN. In *Proceedings of the Australasian Computer Science Week Multiconference*, page 36, 2016.

[60] Yu Sasaki, Yosuke Todo, Kazumaro Aoki, Yusuke Naito, Takeshi Sugawara, Yumiko Murakami, Mitsuru Matsui, and Shoichi Hirose. Minalpher v1.1. Submission to CAESAR, 2015. https://competitions.cr.yp.to/round2/minalpherv11.pdf.

[61] Willem Schroé, Bart Mennink, Elena Andreeva, and Bart Preneel. Forgery and Subkey recovery on CAESAR candidate iFeed. In *SAC*, volume 9566 of *LNCS*, pages 197–204. Springer, 2015.

[62] Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. Piccolo: an ultra-lightweight blockcipher. In *CHES 2011*, pages 342–357, 2011.

[63] Tomoyasu Suzaki, Kazuhiko Minematsu, Sumio Morioka, and Eita Kobayashi. TWINE: a lightweight block cipher for multiple platforms. In *SAC 2012*, pages 339–354, 2012.
[64] Serge Vaudenay. Decorrelation: a theory for block cipher security. *J. Cryptol.*, 16(4):249–286, 2003.
[65] Hongjun Wu. ACORN: A Lightweight Authenticated Cipher (v3). Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/acornv3.pdf.
[66] Hongjun Wu and Tao Huang. The JAMBU Lightweight Authentication Encryption Mode (v2.1). Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/jambuv21.pdf.
[67] Hongjun Wu and Bart Preneel. AEGIS: A Fast Authenticated Encryption Algorithm (v1.1). Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/aegisv11.pdf.
[68] Panasayya Yalla and Jens-Peter Kaps. Evaluation of the CAESAR hardware API for lightweight implementations. In *International Conference on ReConFigurable Computing and FPGAs, ReConFig 2017, Cancun, Mexico, December 4–6, 2017*, pages 1–6. IEEE, 2017.
[69] Liting Zhang, Wenling Wu, Han Sui, and Peng Wang. iFeed[AES] v1. Submission to CAESAR, 2014. https://competitions.cr.yp.to/round1/ifeedaesv1.pdf.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.