Blockcipher-based Authenticated Encryption: How Small Can We Go? ⋆

Avik Chakraborti[1], Tetsu Iwata[2], Kazuhiko Minematsu[3], and Mridul Nandi[4]

[1] NTT Secure Platform Laboratories, Japan, chakraborti.avik@lab.ntt.co.jp
[2] Nagoya University, Japan, iwata@cse.nagoya-u.ac.jp
[3] NEC Corporation, Japan, k-minematsu@ah.jp.nec.com
[4] Applied Statistics Unit, Indian Statistical Institute, Kolkata, mridul.nandi@gmail.com

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. When instantiated with AES-128, COFB achieves only a few more than 1000 Look-Up-Tables (LUTs) while maintaining almost the same level of provable security 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. 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.

⋆ A preliminary version of this paper was presented at CHES 2017 [23].

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 [19] proposed in 2007. Since then, many have been proposed in the last decade, such as KATAN [22], LED [33], PICCOLLO [58], PRINCE [21] and TWINE [59]. SIMON and SPECK [15] are proposed by NSA in 2014. More recent designs are SKINNY (which is a tweakable blockcipher [16]) and GIFT [13, 14].

The other approach is to use standard AES implemented in a tiny, serialized core [47], where the latter is shown to be effective for various schemes including popular CCM [5] or OCB [40] modes, as shown in [20] and [12]. 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 [62] is considered to be one of the most relevant mode 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[‡], and the security claim about the confidentiality in the nonce misuse scenario was shown to be flawed [50]. 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 [36, 37] 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 have also been addressed by NIST's lightweight cryptography project [44], 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.

[‡] The authenticity result was briefly presented in the latest specification [62].

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 [40], 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 [43]) using a blockcipher. We could use existing tweakable blockcipher mode for this purpose, e.g. XEX [51] 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 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 requires 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 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 Sect. 4 and 5 have been described at the proceedings version of our paper in CHES 2017 [23].

Table 1. Comparison of AE modes, using an n-bit blockcipher with k-bit keys. An inverse-free mode is a mode that does not need the blockcipher inverse (decryption) function for both encryption and decryption. For JAMBU, the authenticity bound was briefly presented in [62].

Scheme	State Size	Rate	Parallel	Inverse-Free	Sec. Proof	Ref
COFB	$1.5n + k$	1	No	Yes	Yes	This work
JAMBU	$1.5n + k$	1/2	No	Yes	Partial	[62]
CLOC/ SILC	$2n + k$	1/2	No	Yes	Yes	[36, 37]
iFEED	$3n + k$	1	Only for Enc	Yes	Flawed [57]	[64]
OCB	$\geq 3n + k$	1	Yes	No	Yes	[40, 51, 52]

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 [13, 14]) 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 [23, 24]. COFB[AES] shows the impressive performance figures of COFB both for size and 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 [23]. We have to warn that this is a rough comparison ignoring differences in several implementation factors (see Sect. 6). 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 [7] 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:

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

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

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

where $z = \lceil |Z|/n \rceil$, $|Z[i]| = n$ for all $i < z$ and $1 \leq |Z[z]| \leq n$ such that $Z = (Z[1] \,\|\, Z[2] \,\|\, \cdots \,\|\, Z[z])$. 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 $Z[a..b]$. For integers $a \leq b$, we write $[a..b]$ 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

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

where $(X[1], X[2], \ldots, X[x]) \xleftarrow{1} X$ and thus $X[1..|Y|]$ denotes the first $|Y|$ bits of X. 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 $i_1, \ldots, i_r \in [1..s]$ 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.

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 never repeats at 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 $\mathbf{Adv}_{\mathcal{E}}^{\mathrm{priv}}(q, \sigma, t)$, 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 $\mathbf{Adv}_{\mathcal{E}}^{\mathrm{auth}}(\mathcal{A}) = \Pr[\mathcal{A}^{\mathcal{E}_K, \mathcal{D}_K} \text{ forges}]$, and we write

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

to denote the maximum forging advantage for all adversaries running in time t, making q encryption and q_f decryption queries with total number of queried blocks being at most σ 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 [31, 53]. 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

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

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

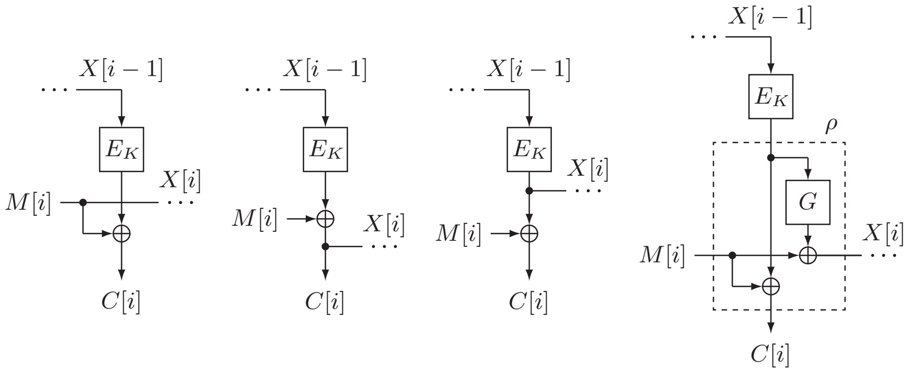

Fig. 3.1. Different types of feedback modes. We introduce the last feedback mode (called the combined feedback mode) in our construction.

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. 3.1. 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, 36, 37, 64]. The fourth mode, which uses additional (linear) operation $G : \mathcal{B} \to \mathcal{B}$, 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: (i) previous output $E_K(X[i-1])$, (ii) plaintext $M[i]$, and (iii) 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 mode in Fig. 3.1). Similarly, it is called ciphertext feedback if X depends only on C in the function ρ' (e.g., the second mode in Fig. 3.1). 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

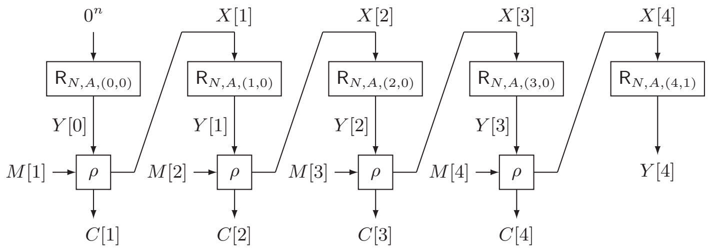

Fig. 3.2. iCOFB: It is based on a tweakable random function $\mathsf{R}_{N,A,(a,b)}$ and a feedback function ρ. The diagram shows how the tag and ciphertext computed for a three complete blocks message.

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. 3.3 and illustrated in Fig. 3.2. 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 assume that the last message 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 non negative 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 non negative integers (i, j) is called a position-tweak.

3.2 The feedback function ρ

The function $\rho : \mathcal{B} \times \mathcal{B} \to \mathcal{B} \times \mathcal{B}$ in the encryption algorithm is called a *feedback function*. The function ρ should be chosen in a way such that there exists a function ρ' (as used in the decryption algorithm) for which decryption algorithm correctly decrypts. In other words, we need an appropriate condition on ρ for the correctness of the encryption algorithm. A necessary and sufficient condition for $\rho : \mathcal{B} \times \mathcal{B} \to \mathcal{B} \times \mathcal{B}$ is the following: there exists a function $\rho' : \mathcal{B} \times \mathcal{B} \to \mathcal{B} \times \mathcal{B}$ such that Eq. (2) holds.

It is easy to see that for such a 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

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. 3.3. Encryption and decryption algorithms of iCOFB AE-mode. Here $M \in \mathcal{B}^m$, $C \in \mathcal{B}^c$ for some $m, c \geq 1$ and $\rho, \rho' : \mathcal{B}^2 \to \mathcal{B}^2$. The choices of these functions are described in Sect. 3.2.

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. 3.3 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'). $\square$

Let ρ be a linear feedback function satisfying Eq. (2) (equivalently $E_{2,2}$ is invertible with the above matrix representation). 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 $D_{2,2} = E_{2,2}^{-1}$. Throughout the paper we assume that $E_{2,2}$ is invertible.

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. 3.1). 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* (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. 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 encryption queries and q_f forging attempts having at most ℓ_f many blocks, we have*

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

Proof. We consider an adversary $\mathcal{A}$ which make q nonce-respecting encryption queries (A_i, N_i, M_i) and receives (C_i, T_i), $1 \le i \le q$, and makes q_f decryption queries $(N_i^*, A_i^*, C_i^*, T_i^*)$, $1 \le i \le q_f$. The intermediate variable Z appeared in the both encryption and decryption algorithms are represented by $Z_i[j]$ for the j-th computation of the i-th query, where Z can be A, M, C, X, Y and t (recall that t is a position-tweak). 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 $C_1, \ldots, C_q$ are uniformly and independently distributed and this would be true provided $Y_1, \ldots, Y_q$ 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, $(N_i, A_i, t_i[j], X_i[j])$ 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 prove it in different cases of forging attempt.

Case $(N^*, A^*) = (N_i, A_i)$**.**
W.o.l.g. we assume that $i = 1$. Let p be the length of the largest common prefix of $((C_1[1], t_1[1]), \ldots, (C_1[m_1], t_1[m_1]))$ and $((C^*[1], t^*[1]), \ldots, (C^*[m^*], t^*[m^*]))$. From the definition of tweak $t[\cdot]$, it is easy to see that $p < \min\{m_1, m^*\}$. So, we have

$$Y_1[p] = Y^*[p], \quad (C_1[p+1], t_1[p+1]) \neq (C^*[p+1], t^*[p+1]).$$

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

For the time being let us assume that this claim is true. So $Y^*[p+1]$ is uniformly distributed given the values obtained so far. 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 $1 - \frac{m^*}{2^n}$. So the forging probability is at most $\frac{(m^*+1)}{2^n}$ for a single attempt.

Case $(N^*, A^*) \neq (N_i, A_i)$ **for all** i**.**
In this case the first tweaked input $(N^*, A^*, t^*[0], 0^n)$ is fresh. We can similarly apply the previous argument to claim that the last tweaked input remains fresh with probability at least $1 - \frac{m^*}{2^n}$. So in this case also, the forging probability is at most $\frac{(m^*+1)}{2^n}$.

In the case of q_f forging attempts, the success probability is at most $\frac{q_f(\ell_f+1)}{2^n}$ (from definition, $m^* \le \ell_f$). This completes the proof, and it remains to show the proof of the claim.

Proof (of Claim). We prove this in two sub-cases. We first note that for all $i \neq p+1$, $t_1[i] \neq t^*[p+1]$ and so it would be sufficient to show that $(t_1[p+1], X_1[p+1]) \neq (t^*[p+1], X^*[p+1])$. If $C_1[p+1] = C^*[p+1]$ then $X_1[p+1] = X^*[p+1]$ but $t_1[p+1] \neq t^*[p+1]$. Similarly, when $C_1[p+1] \neq C^*[p+1]$, by **A1** condition, the next tweaked inputs are distinct.

□

Remark 1. We would like to note the one of key argument in the proof. It says that whenever we obtain a fresh tweaked input, with high probability the last tweaked input remains fresh. So it would be sufficient to identify a position in which the tweaked input for the forging attempt is fresh with high probability. In the above proof for the idealized version, the position is $(p+1)$ and for this position, tweaked input is fresh with probability one. For our main construction, the freshness occurs with high probability instead of probability one. However, the position will be determined in exactly the same way as we did here.

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:

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

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

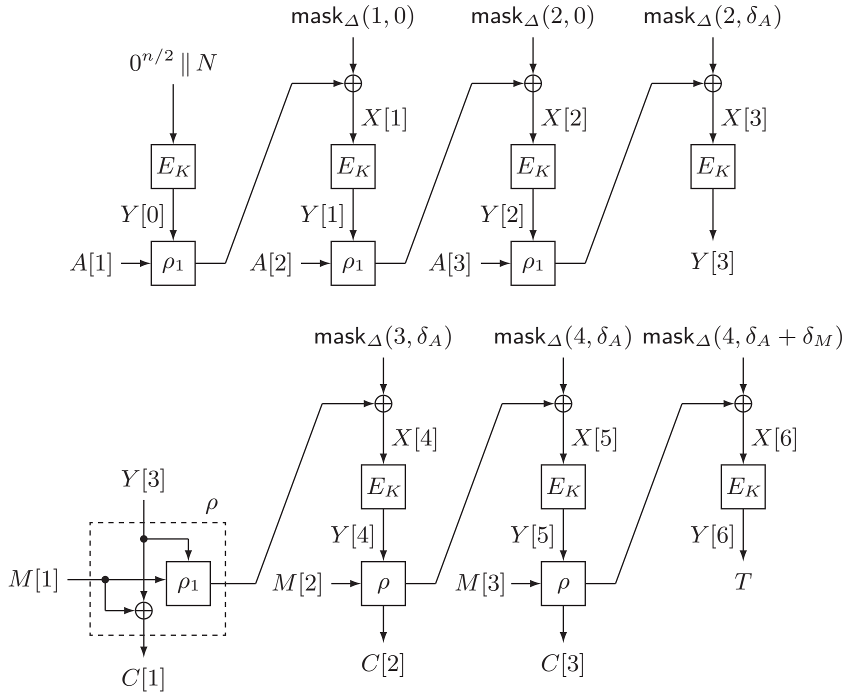

Fig. 4.1. Encryption of COFB for 3-block associated data and plaintext.

primitive polynomial defining the field, we choose the lexicographically first one, that is, $p(x) = \mathtt{x}^{64} + \mathtt{x}^4 + \mathtt{x}^3 + \mathtt{x} + 1$ following [6, 35]. Rogaway [51] showed that for all $(a, b) \in \{0, \ldots, 2^{51}\} \times \{0, \ldots, 2^{10}\}$, the values of $\alpha^a \cdot (1 + \alpha)^b$ are distinct. If we follow the notations of [51], 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 $(a, b) \in \{0, \ldots, L\} \times \{0, \ldots, 4\}$. 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 $\{0, \ldots, L\} \times \{0, \ldots, 4\}$ (as described above), $c \in \{0,1\}^{n/2}$ and a random $n/2$ bit string Δ, we have*

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

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

Similar masking functions are frequently used in other modes, such as [9, 45, 51], 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 [51], 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 $G(Y) = (Y[2], Y[3], Y[4], Y[4] \oplus Y[1])$. 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

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

Note that when $(X, M) = \rho'(Y, C)$ then $X = (G \oplus I) \cdot Y \underline{\oplus} C$. 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.

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

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

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 non-negative integers and $\mathcal{X}$ be some non-empty set. We say that a function $f : \mathcal{X} \to (\mathcal{B} \times \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0})^+$ 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.

Algorithm Mask-Gen(K, N)

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

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

$(\Delta, Y[0]) \leftarrow \text{Mask-Gen}(K, N)$
$(A[1], \ldots, A[a]) \xleftarrow{n} A$
$(M[1], \ldots, M[m]) \xleftarrow{n} M$
for $i = 1$ **to** $a - 1$
  $\Delta \leftarrow 2\Delta$
  $X[i] \leftarrow (A[i] \oplus G \cdot Y[i-1]) \overline{\oplus} \Delta$
  $Y[i] \leftarrow E_K(X[i])$
if $|A[a]| = n$ **then** $\Delta \leftarrow 3\Delta$
else $\Delta \leftarrow 3^2\Delta$
$X[a] \leftarrow (\overline{A[a]} \oplus G \cdot Y[a-1]) \overline{\oplus} \Delta$
$Y[a] \leftarrow E_K(X[a])$
for $i = 1$ **to** $m - 1$
  $X[i+a] \leftarrow (M[i] \oplus G \cdot Y[i+a-1]) \overline{\oplus} \Delta$
  $Y[i+a] \leftarrow E_K(X[i+a])$
  $C[i] \leftarrow Y[i+a-1] \oplus M[i]$
  if $i < m - 1$ **then** $\Delta \leftarrow 2\Delta$
if $|M[m]| = n$ **then** $\Delta \leftarrow 3\Delta$
else $\Delta \leftarrow 3^2\Delta$
$X[a+m] \leftarrow (\overline{M[m]} \oplus G \cdot Y[a+m-1]) \overline{\oplus} \Delta$
$C[m] \leftarrow Y[a+m-1] \underline{\oplus} M[m]$
$T \leftarrow E_K(X[a+m])$
return (C, T)

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

$(\Delta, Y[0]) \leftarrow \text{Mask-Gen}(K, N)$
$(A[1], \ldots, A[a]) \xleftarrow{n} A$
$(C[1], \ldots, C[c]) \xleftarrow{n} C$
for $i = 1$ **to** $a - 1$
  $\Delta \leftarrow 2\Delta$
  $X[i] \leftarrow (A[i] \oplus G \cdot Y[i-1]) \overline{\oplus} \Delta$
  $Y[i] \leftarrow E_K(X[i])$
if $|A[a]| = n$ **then** $\Delta \leftarrow 3\Delta$
else $\Delta \leftarrow 3^2\Delta$
$X[a] \leftarrow (\overline{A[a]} \oplus G \cdot Y[a-1]) \overline{\oplus} \Delta$
$Y[a] \leftarrow E_K(X[a])$
for $i = 1$ **to** $c - 1$
  $X[i+a] \leftarrow (C[i] \oplus Y[i+a-1] \oplus G \cdot Y[i+a-1]) \overline{\oplus} \Delta$
  $M[i] \leftarrow Y[i+a-1] \oplus C[i]$
  $Y[i+a] \leftarrow E_K(X[i+a])$
  if $i < c - 1$ **then** $\Delta \leftarrow 2\Delta$
if $|C[a]| = n$ **then** $\Delta \leftarrow 3\Delta$
else $\Delta \leftarrow 3^2\Delta$
$X[a+c] \leftarrow (\overline{C[c]} \oplus Y[a+c-1] \oplus G \cdot Y[a+c-1]) \overline{\oplus} \Delta$
$M[c] \leftarrow Y[a+c-1] \underline{\oplus} C[c]$
$T' \leftarrow E_K(X[a+c])$
$M \leftarrow (M[1], \ldots, M[c])$
if $T' = T$ **then return** M
else return $\perp$

Fig. 4.2. The encryption and decryption algorithms of COFB.

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. 4.2, where α and $(1 + \alpha)$ in Eq. (3) are written as 2 and 3. See also Fig. 4.1. The encryption and decryption algorithms are denoted by $\mathsf{COFB}\text{-}\mathcal{E}_K$ and $\mathsf{COFB}\text{-}\mathcal{D}_K$. 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. 4.2, since they are introduced for the proof. An equivalent presentation using them is presented in Fig. 5.1.

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

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

where $f(N,A)$ is the function that simulates the associated data phase and outputs $Y[a]$ (Line 1–11, Fig. 4.2, 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).

$$\mathbf{Adv}^{\mathrm{AE}}_{\mathsf{COFB}}((q,q_f),(\sigma,\sigma_f),t) \le \mathbf{Adv}^{\mathrm{prp}}_{\mathrm{AES}}(q',t') + \frac{0.5(q')^2}{2^n} + \frac{4\sigma+0.5nq_f}{2^{n/2}} + \frac{q_f+(q+\sigma+\sigma_f)\cdot\sigma_f}{2^n},$$

where $q' = q+q_f+\sigma+\sigma_f$, which corresponds to the total number of blockcipher calls through the game, and $t' = t+O(q')$.

Proof. Without loss of generality, we can assume $q' \le 2^{\frac{n}{2}-1}$, since otherwise the bound obviously holds as the right hand side becomes more than one. 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 [17]). Then what we need is a bound for COFB using R, denoted by $\mathsf{COFB\text{-}R}$. That is, we prove

$$\mathbf{Adv}^{\mathrm{AE}}_{\mathsf{COFB\text{-}R}}((q,q_f),(\sigma,\sigma_f),\infty) \le \frac{4\sigma+0.5nq_f}{2^{n/2}} + \frac{q_f+(q+\sigma+\sigma_f)\cdot\sigma_f}{2^n}. \quad (5)$$

For $i=1,\dots,q$, we write (N_i,A_i,M_i) and (C_i,T_i) to denote the i-th encryption query and response. Here, $A_i = (A_i[1],\dots,A_i[a_i])$, $M_i = (M_i[1],\dots,M_i[m_i])$, and $C_i = (C_i[1],\dots,C_i[m_i])$. Let $\ell_i = a_i+m_i$, which denotes the total input block length for the i-th encryption query. We write $X_i[j]$ (resp. $Y_i[j]$) for $i=1,\dots,q$ and $j=0,\dots,\ell_i$ to denote the j-th input (resp. output) of the internal R invoked at the i-th encryption query, where the order of invocation follows the specification shown in Fig. 4.2. We remark that $X_i[0] = 0^{n/2}\|N_i$ and $Y_i[\ell_i] = T_i$ for all $i=1,\dots,q$. Similarly, we write Δ_i to denote $Y_i^2[0]\|Y_i^3[0]$ where $Y_i^1[0]\|\cdots\|Y_i^4[0] \xleftarrow{n/4} Y_i[0]$.

We introduce the following relaxations in the game, which only gain the advantage. First, after completing all queries and forging attempts (i.e. decryption queries), let the adversary learn all the Y-values for all encryption queries only. We remark that any X-values computed at the message processing phase (not the AD processing phase) of the i-th encryption query are immediately determined by the i-th query-response tuple, $(N_i, A_i, M_i, C_i, T_i)$ and Y_i values from the property of feedback function, and Δ-values (it is a part of $Y[0]$).

In case of the ideal oracle, all these variables corresponding to Y will be chosen uniformly and independently, where at the plaintext encryption phase $Y_i[j]$ is randomly chosen and used to determine $C_i[j]$ as $C_i[j] = Y_i[j-1] \underline{\oplus} M_i[j]$, and at AD processing phase it is a dummy and has no influence to the response (C_i, T_i). For decryption queries, the ideal oracle always returns $\perp$ (here we assume that the adversary makes only fresh queries).

Coefficients-H Technique. We outline the Coefficients-H technique developed by Patarin, which serves as a convenient tool for bounding the advantage (see [49, 60]). 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, $\mathcal{V}_{\text{good}} \subseteq \mathcal{V}$, 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 $\mathcal{V}_{\text{good}}$ is at least $1 - \epsilon_1$.
2. For any view $\tau \in \mathcal{V}_{\text{good}}$, 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) [60]. Now we proceed with the proof of Theorem 2 by defining certain $\mathcal{V}_{\text{good}}$ for our games, and evaluating the bounds, ϵ_1 and ϵ_2.

Views. In our case, a view τ is defined by the following tuple:

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

where $Z^*_{i'}$ 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 $(N^*_{i'}, A^*_{i'}, C^*_{i'}, T^*_{i'})$. Note that Y_i denotes $(Y_i[0], \ldots, Y_i[\ell_i]) = Y_i[0..\ell_i]$, where $\ell_i = a_i + m_i$, and a_i (resp. m_i) denotes the block length of A_i (resp. M_i). Here we implicitly use the fact that given a complete block $M_i[j]$, the mapping from $Y_i[j]$ to $C_i[j]$ is bijective and hence keeping those $Y_i[j]$ values instead of $C_i[j]$ is sufficient. Similarly we define $c^*_{i'}$ and $a^*_{i'}$, and write $\ell^*_{i'} = a^*_{i'} + c^*_{i'}$.

Let $(L_i[j], R_i[j]) \xleftarrow{n/2} X_i[j]$ for all $i \in [1..q]$ and $j \in [1..\ell_i]$. 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 $N_j = N^*_i$. If there is no such j, we define $p_i = -1$. Since Fmt is prefix-free, it holds that $p_i < \min\{\ell^*_i, \ell_j\}$. We observe that p_i is unique for all $i = 1, \ldots, q_f$, as there is at most one encryption query that uses the same nonce as N^*_i.

Bad Views. Now we define a bad view. The complement of the set of bad views is defined to be the set of good views. A view is called bad if one of the following events occurs:

B1: $L_i[j] = 0^{n/2}$ for some $i \in [1..q]$ and $j > 0$.
B2: $X_i[j] = X_{i'}[j']$ for some $(i, j) \neq (i', j')$ where $j, j' > 0$.
B3: $\mathsf{mcoll}(R) > n/2$, where R is the tuple of all $R_i[j]$ values. Recall that $(L_i[j], R_i[j]) \xleftarrow{n/2} X_i[j]$.
B4: $X^*_i[p_i + 1] = X_{i_1}[j_1]$ for some i, i_1, j_1 with p_i as defined above. Note that when $p_i \geq 0$, $X^*_i[p_i + 1]$ is determined from the values of Y.
B5: For some $Z^*_i \neq \perp$. This clearly cannot happen for the ideal oracle case.

We add some intuitions on these events. When **B1** does not hold, then $X_i[j] \neq X_{i'}[0]$ 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 $X_i[j]$ 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. Finally, when **B4** does not hold, then the $(p_i + 1)$-st input for the i-th forging attempt will be fresh with a high probability and so all the subsequent inputs will remain fresh with a high probability.

A view is called good if none of the above events hold. Let $\mathcal{V}_{\text{good}}$ be the set of all such good views. 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.

$$\Pr_{\mathsf{ideal}}[\tau \notin \mathcal{V}_{\text{good}}] \leq \frac{4\sigma + 0.5nq_f}{2^{n/2}}.$$

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 $(Y^1_i[j], Y^2_i[j], Y^3_i[j], Y^4_i[j]) \xleftarrow{n/4} Y_i[j]$. Similarly, we denote $(M^1_i[j], M^2_i[j]) \xleftarrow{n/2} M_i[j]$.

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

$$(Y_i^2[j-1] \| Y_i^3[j-1]) \oplus (\alpha^a \cdot (1+\alpha)^b \cdot \Delta_i) \oplus M_i^1[j]$$

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

(2) $\Pr[\mathbf{B2}] \le \sigma/2^{n/2}$: For any $(i,j) \ne (i',j')$ with $j, j' \ge 1$, the equality event $X_i[j] = X_{i'}[j']$ has a probability at most 2^{-n} since this event is a non-trivial linear equation on $Y_i[j-1]$ and $Y_{i'}[j'-1]$ and they are independent to each other. Note that $\sigma^2/2^n \le \sigma/2^{n/2}$ as we are estimating probabilities.

(3) $\Pr[\mathbf{B3}] \le 2\sigma/2^{n/2}$: 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

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

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

(4) $\Pr[\mathbf{B4} \wedge \mathbf{B1}^c \wedge \mathbf{B3}^c] \le 0.5nq_f/2^{n/2}$: We fix some i and want to bound the probability $\Pr[X_i^*[p_i+1] = X_{i_1}[j_1] \wedge \mathbf{B1}^c \wedge \mathbf{B3}^c]$ for some i_1, j_1. If $p_i = -1$ (i.e., N_i^* does not appear in encryption queries), then N_i^* is fresh as left $n/2$ bits of all $X_i[j]$ is non-zero for all $j > 0$ (since we also consider $\mathbf{B1}$ does not hold). So the probability is zero. Now we consider $p_i \ge 0$. The event $\mathbf{B3}^c$ implies that at most $n/2$ possible values of (i_1, j_1) are possible for which $X_i^*[p_i+1] = X_{i_1}[j_1]$ can hold. Fix any such (i_1, j_1). Now it is sufficient to bound the probability for equality for the left $n/2$ bits. We first consider the case where $j_1 = p_i + 1$. Now from the definition of p_i, $(C_i^*[p_i+1], \mathsf{t}_i^*[p_i+1]) \ne (C_{i_1}[p_i+1], \mathsf{t}_{i_1}[p_i+1])$. If $\mathsf{t}_i[p_i+1] = \mathsf{t}_{i_1}[p_i+1]$ then the bad event cannot hold with probability one. Otherwise, we obtain a non-trivial linear equation in Δ_{i_1} and apply Lemma 2, and we also use the fact that $G + I$ is non singular. A similar argument holds for the other choices of j_1. Therefore, the probability for the atomic case is at most $2^{-n/2}$, and because we have at most $q_f \cdot n/2$ chances, $\Pr[\mathbf{B4} \wedge \mathbf{B1}^c \wedge \mathbf{B3}^c]$ is at most $(n/2) \cdot q_f \cdot 1/2^{n/2}$.

Summarizing, we have

$$\begin{aligned}\Pr_{\mathsf{ideal}}[\tau \notin \mathcal{V}_{\text{good}}] &\le \Pr[\mathbf{B1}] + \Pr[\mathbf{B2}] + \Pr[\mathbf{B3}] + \Pr[\mathbf{B4} \wedge \mathbf{B1}^c \wedge \mathbf{B3}^c] \\ &\le \frac{\sigma}{2^{n/2}} + \frac{\sigma}{2^{n/2}} + \frac{2\sigma}{2^{n/2}} + \frac{0.5nq_f}{2^{n/2}} = \frac{4\sigma + 0.5nq_f}{2^{n/2}},\end{aligned}$$

which concludes the proof. □

Lower Bound of $\mathsf{ip}_{\mathsf{real}}(\tau)$. We consider the ratio of $\mathsf{ip}_{\mathsf{real}}(\tau)$ and $\mathsf{ip}_{\mathsf{ideal}}(\tau)$. In this paragraph we assume that all the probability space, except for $\mathsf{ip}_{\mathsf{ideal}}(*)$, is defined over the real game. We fix a good view

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

where $Z^*_{i'} = \bot$. We separate τ into

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

and we first see that for a good view τ, $\mathsf{ip}_{\mathsf{ideal}}(\tau)$ equals to $1/2^{n(q+\sigma)}$.

Now we consider the real case. Since **B1** and **B2** do not hold with τ, all inputs of the random function inside τ_e are distinct, which implies that the released Y-values are independent and uniformly random. The variables in τ_e are uniquely determined given these Y-values, and there are exactly $q+\sigma$ distinct input-output of R. Therefore, $\Pr[\tau_e]$ is exactly $2^{-n(q+\sigma)}$.

We next evaluate

$$\mathsf{ip}_{\mathsf{real}}(\tau) = \Pr[\tau_e, \tau_d] = \Pr[\tau_e] \cdot \Pr[\tau_d|\tau_e] = \frac{1}{2^{n(q+\sigma)}} \cdot \Pr[\tau_d|\tau_e]. \tag{6}$$

We observe that $\Pr[\tau_d|\tau_e]$ equals to $\Pr[\bot_{\mathsf{all}}|\tau_e]$, where $\bot_{\mathsf{all}}$ denotes the event that $Z^*_i = \bot$ for all $i = 1,\ldots,q_f$, as other variables in τ_d are determined by τ_e.

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

Now we have $\Pr[\bot_{\mathsf{all}}|\tau_e] = 1 - \Pr[(\bot_{\mathsf{all}})^c|\tau_e]$, and we also have $\Pr[(\bot_{\mathsf{all}})^c|\tau_e] = \Pr[(\bot_{\mathsf{all}})^c, \eta|\tau_e] + \Pr[(\bot_{\mathsf{all}})^c, \eta^c|\tau_e]$. Here, $\Pr[(\bot_{\mathsf{all}})^c, \eta|\tau_e]$ is the probability that at least one T^*_i for some $i = 1,\ldots,q_f$ is correct as a guess of $Y^*_i[\ell^*_i]$. Here $Y^*_i[\ell^*_i]$ is completely random from η, hence using the union bound we have

$$\Pr[(\bot_{\mathsf{all}})^c, \eta|\tau_e] \le \frac{q_f}{2^n}.$$

For $\Pr[(\bot_{\mathsf{all}})^c, \eta^c|\tau_e]$ which is at most $\Pr[\eta^c|\tau_e]$, 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 the all X-values in τ_e and all X^*-values in τ_d, as in the same manner to the collision analysis of e.g., CBC-MAC using R. For each i-th decryption query, the number of bad pairs is at most $(q+\sigma+\ell^*_i)\cdot\ell^*_i \le (q+\sigma+\sigma_f)\cdot\ell^*_i$. Therefore, the total number of bad pairs is $\sum_{1\le i\le q_f}(q+\sigma+\sigma_f)\cdot\ell^*_i \le (q+\sigma+\sigma_f)\cdot\sigma_f$, and we have

$$\Pr[(\bot_{\mathsf{all}})^c, \eta^c|\tau_e] \le \frac{(q+\sigma+\sigma_f)\cdot\sigma_f}{2^n}.$$

Module Mask-Gen(K, N)

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

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

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

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

$(\Delta, Y[0]) \leftarrow$ Mask-Gen(K, N)
$(A[1], \ldots, A[a]) \xleftarrow{n} A$
$(C[1], \ldots, C[c]) \xleftarrow{n} C$
$\ell \leftarrow a + c$
$((B[1], \mathsf{t}[1]), \ldots, (B[\ell], \mathsf{t}[\ell])) \leftarrow \mathsf{Fmt}(A, C)$
for $i = 1$ **to** ℓ
  if $i \le a$ **then**
    $X[i] \leftarrow (B[i] \oplus G \cdot Y[i-1]) \overline{\oplus}\, \mathsf{mask}_\Delta(\mathsf{t}[i])$
  else $X[i] \leftarrow (B[i] \oplus Y[i-1] \oplus G \cdot Y[i-1]) \overline{\oplus}\, \mathsf{mask}_\Delta(\mathsf{t}[i])$
  $Y[i] \leftarrow E_K(X[i])$
for $i = 1$ **to** c
  $M[i] \leftarrow Y[i+a-1] \underline{\oplus}\, C[i]$
$M \leftarrow (M[1], \ldots, M[c])$
$T' \leftarrow Y[\ell]$
if $T' = T$ **then return** M
else return $\perp$

Fig. 5.1. A presentation of COFB using Fmt function. This is equivalent to Fig. 4.2.

Combining all, we have

$$\begin{aligned}
\mathsf{ip}_{\mathsf{real}}(\tau) &= \frac{1}{2^{n(q+\sigma)}} \cdot \Pr[\tau_d | \tau_e] = \mathsf{ip}_{\mathsf{ideal}}(\tau) \cdot \Pr[\perp_{\mathsf{all}} | \tau_e] \\
&\ge \mathsf{ip}_{\mathsf{ideal}}(\tau) \cdot (1 - (\Pr[(\perp_{\mathsf{all}})^c, \eta | \tau_e] + \Pr[(\perp_{\mathsf{all}})^c, \eta^c | \tau_e])) \\
&\ge \mathsf{ip}_{\mathsf{ideal}}(\tau) \cdot \left(1 - \frac{q_f + (q + \sigma + \sigma_f) \cdot \sigma_f}{2^n}\right).
\end{aligned}$$

□

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 parallelizing the internal components is not needed. In this respect, COFB's small state size and completely serial operation is 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

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

lightweight blockcipher proposed by Banik et al. [13]. It employs a structure similar to PRESENT [19] 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 Architecture

We describe the implementation details of both COFB[AES] and COFB[GIFT]. 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. 6.2. 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 $BC \in \{\text{AES-128}, \text{GIFT-128}\}$. 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. 6.1. 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 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.

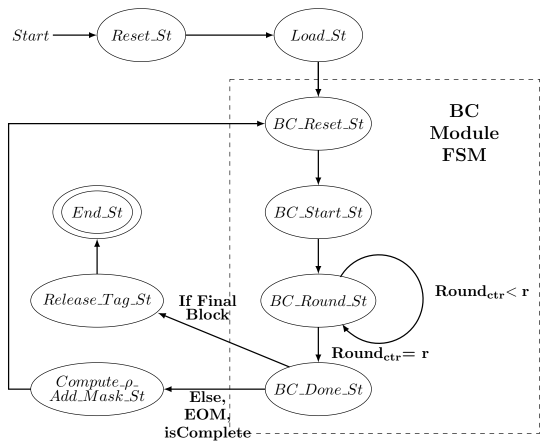

Fig. 6.1. FSM for $\mathsf{COFB}[BC]$ Hardware Implementation

Basic Implementation. We describe a basic flow of our implementation, which generally follows the pseudocode of Fig. 4.2. Prior to the initialization, **State** register is loaded with $0^{64} \| N$. Once **State** register is initialized, the initialization process starts by encrypting the nonce $(0^{64} \| N)$ with BC. Then, 64 bits of the encrypted nonce is chopped by the "chop" function as in Fig. 6.2, 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 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.

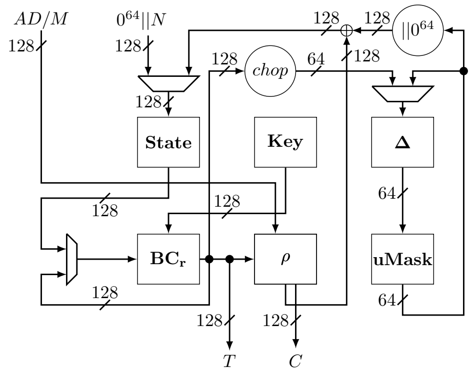

Fig. 6.2. Hardware Circuit Diagram

Table 4. FPGA implementation results of COFB[AES]

Platform	Slice Registers	LUTs	Slices	Frequency (MHz)	Throughput (Gbps)	Mbps/LUT	Mbps/Slice
Virtex 6	594	1051	449	267.20	2.85	2.71	6.35
Virtex 7	593	1440	564	274.84	2.93	2.03	5.19

6.3 Implementation Results

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 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 [23, 24] 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,

Table 5. FPGA implementation results of COFB[GIFT]

Platform	Slice Registers	LUTs	Slices	Frequency (MHz)	Throughput (Gbps)	Mbps/LUT	Mbps/Slice
Virtex 6	342	771	355	612.91	1.91	2.48	5.51
Virtex 7	342	771	316	712.99	2.23	2.89	6.62

BP, ARK and ARC, $UKEY$. operations. SN module applies a 4-bit sbox to each of the 4-bit nibbles of the 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 5.

6.4 Hardware Flexibility of the COFB Design

COFB is itself very lightweight and it uses a few operations other than the blockcipher computations. Below in Table 6 and 7, 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.

Table 6. COFB[AES]: Area utilization by modules in Virtex 6

Modules	Slices	LUTs
Total	449	1051
AES	311	657
Others	138	394

Table 7. COFB[GIFT]: Area utilization by modules in Virtex 6

Modules	Slices	LUTs
Total	355	771
GIFT	155	346
Others	200	425

6.5 Benchmarking with ATHENa Database

We compare our implementation of COFB with the results published in ATHENa Database [2], taking Virtex 6 and Virtex 7 as our target platforms. We first warn that this is a rough comparison. Here, we ignore the overhead to support the GMU 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. We acknowledge that supporting GMU API will require some additional overhead to the current figures of COFB. Nevertheless, we think the current figures of COFB suggest that small hardware implementations (small for AES-128 and even smaller with GIFT-128) are possible compared with other blockcipher AE modes shown in the table, using the same AES-128 and GIFT-128, even if we add a circuit for supporting GMU API and decryption.

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 [27, 55, 54, 42]. Joltik and JAMBU-SIMON employ lightweight (tweakable) blockciphers allowing smaller implementation than AES, and 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. In Table 8 and 9, we provide the comparison table both Vertex 6 and Vertex 7 platforms.

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 two hardware implementation results for COFB with AES-128 and GIFT-128 blockcipher respectively. 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`.

Table 8. Comparison on Virtex 6 [2]. In the "Primitive" column, SC denotes Stream cipher, (T)BC denotes (Tweakable) blockcipher, and BC-RF denotes the blockcipher's round function.

Scheme	Primitive	LUT	Slices	T'put (Gbps)	Mbps / LUT	Mbps / Slice
ACORN [61]	SC	455	135	3.112	6.840	23.052
AEGIS [63]	BC-RF	7592	2028	70.927	9.342	34.974
AES-COPA [10]	BC	7754	2358	2.500	0.322	1.060
AES-GCM [29]	BC	3175	1053	3.239	1.020	3.076
AES-OTR [46]	BC	5102	1385	2.741	0.537	1.979
AEZ [34]	BC-RF	4597	1246	8.585	0.747	2.756
ASCON [28]	Sponge	1271	413	3.172	2.496	7.680
CLOC [37]	BC	3145	891	2.996	0.488	1.724
DEOXYS [39]	TBC	3143	951	2.793	0.889	2.937
ELmD [26]	BC	4302	1584	3.168	0.736	2.091
JAMBU-AES [62]	BC	1836	652	1.999	1.089	3.067
JAMBU-SIMON [62]	BC (non-AES)	1222	453	0.363	0.297	0.801
Joltik [38]	TBC	1292	442	0.853	0.660	0.826
Ketje [18]	Sponge	1270	456	7.345	5.783	16.107
Minalpher [56]	BC (non-AES)	2879	1104	1.831	0.636	1.659
NORX [11]	Sponge	2964	1016	11.029	3.721	10.855
PRIMATES-HANUMAN [8]	Sponge	1012	390	0.964	0.953	2.472
OCB [41]	BC	4249	1348	3.122	0.735	2.316
SCREAM [32]	TBC	2052	834	1.039	0.506	1.246
SILC [37]	BC	3066	921	4.040	1.318	4.387
Tiaoxin [48]	BC-RF	7123	2101	52.838	7.418	25.149
TriviA-ck [25]	SC	2118	687	15.374	7.259	22.378
COFB[AES]	BC	1051	449	2.850	2.710	6.350
COFB[GIFT]	BC	771	355	1.91	2.48	5.51

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. NIST FIPS 197. Advanced Encryption Standard (AES). *Federal Information Processing Standards Publication*, 197, 2001.
8. 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`.
9. 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.
10. 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`.

Table 9. 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	1373	401	3.852	2.806	9.606
CLOC	3552	1087	3.252	0.478	1.561
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	1125	351	8.718	7.749	24.838
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	3040	910	4.365	1.436	4.796
Tiaoxin	7556	1985	75.776	10.029	38.174
TriviA-ck	2221	684	14.852	6.687	21.713
COFB[AES]	1440	564	2.93	2.03	5.19
COFB[GIFT]	771	316	2.23	2.89	6.62

11. Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. NORX v3.0. Submission to CAESAR. 2016. `https://competitions.cr.yp.to/round3/norxv30.pdf`.
12. Subhadeep Banik, Andrey Bogdanov, and Kazuhiko Minematsu. Low-Area Hardware Implementations of CLOC, SILC and AES-OTR. DIAC 2015.
13. 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 [30], pages 321–345.
14. Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Siang Meng Sim, Yosuke Todo, and Yu Sasaki. GIFT: A small present. *IACR Cryptology ePrint Archive*, 2017:622, 2017.
15. 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.
16. 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.

17. 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.
18. 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`.
19. 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.
20. Andrey Bogdanov, Florian Mendel, Francesco Regazzoni, Vincent Rijmen, and Elmar Tischhauser. ALE: AES-Based Lightweight Authenticated Encryption. In *FSE 2013*, pages 447–466, 2013.
21. 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.
22. 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.
23. Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, and Mridul Nandi. Blockcipher-based authenticated encryption: How small can we go? In Fischer and Homma [30], pages 277–298.
24. Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, and Mridul Nandi. Blockcipher-based authenticated encryption: How small can we go? *IACR Cryptology ePrint Archive*, 2017:649, 2017.
25. Avik Chakraborti and Mridul Nandi. TriviA-ck-v2. Submission to CAESAR. 2015. `https://competitions.cr.yp.to/round2/triviackv2.pdf`.
26. Nilanjan Datta and Mridul Nandi. Proposal of ELmD v2.1. Submission to CAESAR. 2015. `https://competitions.cr.yp.to/round2/elmdv21.pdf`.
27. Prakash Dey, Raghvendra Singh Rohit, and Avishek Adhikari. Full key recovery of ACORN with a single fault. *J. Inf. Sec. Appl.*, 29:57–64, 2016.
28. Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2. Submission to CAESAR. 2016. `https://competitions.cr.yp.to/round3/asconv12.pdf`.
29. 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`.
30. 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.
31. 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.
32. 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`.
33. Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED Block Cipher. In *CHES 2011*, pages 326–341, 2011.
34. 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`.
35. Tetsu Iwata and Kaoru Kurosawa. OMAC: One-Key CBC MAC. In *FSE*, pages 129–153, 2003.
36. Tetsu Iwata, Kazuhiko Minematsu, Jian Guo, and Sumio Morioka. CLOC: Authenticated Encryption for Short Input. In *FSE 2014*, pages 149–167, 2014.
37. 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`.
38. Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. Joltik v1.3. Submission to CAESAR. 2015. `https://competitions.cr.yp.to/round2/joltikv13.pdf`.
39. Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. Deoxys v1.41. Submission to CAESAR. 2016. `https://competitions.cr.yp.to/round3/deoxysv141.pdf`.
40. Ted Krovetz and Phillip Rogaway. The Software Performance of Authenticated-Encryption Modes. In *FSE*, pages 306–327, 2011.
41. Ted Krovetz and Phillip Rogaway. OCB(v1.1). Submission to CAESAR. 2016. `https://competitions.cr.yp.to/round3/ocbv11.pdf`.
42. Frédéric Lafitte, Liran Lerman, Olivier Markowitch, and Dirk Van Heule. SAT-based cryptanalysis of ACORN. *IACR Cryptology ePrint Archive*, 2016:521, 2016.
43. Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable Block Ciphers. In *CRYPTO*, pages 31–46, 2002.
44. 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`.
45. Kazuhiko Minematsu. Parallelizable Rate-1 Authenticated Encryption from Pseudorandom Functions. In *EUROCRYPT*, volume 8441 of *LNCS*, pages 275–292. Springer, 2014.
46. Kazuhiko Minematsu. AES-OTR v3.1. Submission to CAESAR. 2016. `https://competitions.cr.yp.to/round3/aesotrv31.pdf`.
47. 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.
48. Ivica Nikolić. Tiaoxin – 346. Submission to CAESAR. 2016. `https://competitions.cr.yp.to/round3/tiaoxinv21.pdf`.
49. J. Patarin. Etude des Générateurs de Permutations Basés sur le Schéma du D.E.S. Phd Thèsis de Doctorat de l'Université de Paris 6, 1991.
50. Thomas Peyrin, Siang Meng Sim, Lei Wang, and Guoyan Zhang. Cryptanalysis of JAMBU. In *FSE 2015*, pages 264–281, 2015.
51. Phillip Rogaway. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In *ASIACRYPT*, pages 16–31, 2004.
52. 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.
53. Phillip Rogaway and Thomas Shrimpton. A Provable-Security Treatment of the Key-Wrap Problem. In *EUROCRYPT*, pages 373–390, 2006.

54. 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.
55. 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.
56. 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`.
57. 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.
58. 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.
59. Tomoyasu Suzaki, Kazuhiko Minematsu, Sumio Morioka, and Eita Kobayashi. TWINE : A Lightweight Block Cipher for Multiple Platforms. In *SAC 2012*, pages 339–354, 2012.
60. Serge Vaudenay. Decorrelation: A Theory for Block Cipher Security. *J. Cryptology*, 16(4):249–286, 2003.
61. Hongjun Wu. ACORN: A Lightweight Authenticated Cipher (v3). Submission to CAESAR. 2016. `https://competitions.cr.yp.to/round3/acornv3.pdf`.
62. 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`.
63. 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`.
64. Liting Zhang, Wenling Wu, Han Sui, and Peng Wang. iFeed[AES] v1. Submission to CAESAR. 2014. `https://competitions.cr.yp.to/round1/ifeedaesv1.pdf`.