Lightweight Authenticated Encryption Mode of Operation for Tweakable Block Ciphers

Yusuke Naito and Takeshi Sugawara

Mitsubishi Electric Corporation
The University of Electro-Communications

Abstract. The use of a small block length is a common strategy when designing lightweight (tweakable) block ciphers (TBCs), and several 64-bit primitives have been proposed. However, when such a 64-bit primitive is used for an authenticated encryption with birthday-bound security, it has only 32-bit data complexity, which is subject to practical attacks. To employ a short block length without compromising security, we propose PFB, a lightweight TBC-based authenticated encryption with associated data mode, which achieves beyond-birthday-bound security. For this purpose, we extend iCOFB, which is originally defined with a tweakable random function. Unlike iCOFB, the proposed method can be instantiated with a TBC using a fixed tweak length and can handle variable-length data. Moreover, its security bound is improved and independent of the data length; this improves the key lifetime, particularly in lightweight blocks with a small size. The proposed method also covers a broader class of feedback functions because of the generalization presented in our proof. We evaluate the concrete hardware performances of PFB, which benefits from the small block length and shows particularly good performances in threshold implementation.

Keywords: Authenticated encryption · beyond-birthday-bound security · tweakable block cipher · lightweight · threshold implementation

1 Introduction

Driven by a demand for secure connectivity in resource-constrained embedded devices, lightweight cryptography has been actively studied in the last decade. Consequently, several lightweight block ciphers have been proposed [BBI$^+$15, BSS$^+$13, BJK$^+$16, BCG$^+$12, GPPR11, SIH$^+$11, SMMK13], including PRESENT [BKL$^+$07] and CLEFIA [SSA$^+$07] standardized in ISO/IEC 29192-2.

A common strategy for designing a lightweight block cipher is to use a small block length. For example, PRESENT [BKL$^+$07] and PRINCE [BCG$^+$12] support 64-bit block length only. Many other algorithms, such as GIFT [BPP$^+$17] and SKINNY [BJK$^+$16], provide 64-bit options. The small block length contributes to a smaller memory footprint and a shorter round number, which are crucial for a lightweight implementation.

Resource-constrained devices are frequently used in a hostile environment, in which side-channel attack (SCA) [KJJ99] should be considered. Designers face an even more challenging task of realizing an SCA-resistant implementation with limited resources. Researchers have tackled this problem and proposed numerous lightweight and SCA-resistant implementations [NRR06, PMK$^+$11, MPL$^+$11, BJK$^+$16, GJC$^+$17], including the ones protected by threshold implementation (TI) [NRS11]. An advantage of a block cipher with a small block length (i.e., a small state size) becomes even larger with TI, where a shared representation of the state multiplies the memory requirement.

To leverage the benefit of a lightweight block cipher for realizing both confidentiality and integrity, lightweight modes of operation for authenticated encryption with associated data (AEAD) have been actively studied in the past few years, promoted by the CAESAR competition and NIST's move toward standardizing lightweight cryptography [NIS18]. Until now, lightweight and block-cipher-based AEAD modes, such as COFB [CIMN17] and SAEB [NMSS18], have been proposed. However, the short block length of lightweight cryptography can be a challenge in terms of security. The lightweight AEAD modes have security up to the so-called birthday bound. More specifically, the security is ensured up to $O(2^{b/2})$ block-cipher calls when instantiated with a b-bit block cipher. With a 64-bit block cipher, the security is ensured up to 2^{32} block-cipher calls only. It is subject to a practical attack, as demonstrated by the Sweet32 attack [BL16].

The use of an AEAD mode with beyond-birthday-bound (BBB) security is a solution for avoiding the birthday problem. There are block-cipher-based AEAD modes with BBB security, including CHM [Iwa06], CIP [Iwa08], and AEAD modes with CLRW2 [LST12] or r-CLRW [LS13]. However, they are costly, compared to the lightweight AEAD modes, as they require two or more independent universal hash functions. Another solution is to construct a (dedicated) TBC-based AEAD mode. The TBC-based AEAD modes, including TAE [LRW02] (where the procedure of handling associated data is not defined), ΘCB3 [KR11], $\mathbb{OTR}$ [Min14], SCT [PS16], and ZAE [IMPS17], realize better efficiency and security. Especially, TAE, ΘCB3, and SCT have the smallest state in the category of BBB-secure AEAD modes.

1.1 Motivation, Approach, and Problems

Our objective is to design a lightweight BBB-secure and nonce-based AEAD mode, thereby employing a short block length without compromising security. For making the design lightweight, we use the four criteria for lightweight AEAD mentioned in [NMSS18], which are used in designing the block-cipher-based lightweight AEAD mode SAEB, as presented in Table 1 (the design principle originates from Sponge-based schemes [BDPA07, BDPA11]):

- **No extra state**: The AEAD mode does not use any additional memory besides the state and (twea)key registers within the (tweakable) block cipher that can be updated in place.
- **Inverse free**: The AEAD mode does not use decryption call of the (tweakable) block cipher.
- **Linear only**: The AEAD mode needs only linear operations besides the (tweakable) block cipher.[1]
- **Online**: The AEAD mode scans the incoming message only once.

The use of a (dedicated) TBC is a promising approach for designing a lightweight and BBB-secure AEAD mode; however, none of the previous TBC-based AEAD modes, including TAE, ΘCB3, and SCT, satisfy all the lightweight criteria (see Table 1).

Our approach involves designing a (dedicated) TBC-based AEAD mode by extending the idea of iCOFB [CIMN17]. iCOFB shown in Figure 1 is a generalization of COFB by using the tweakable random function (TRF) R having a b-bit output, an arbitrary-length associated data (AD) A, a nonce N, and a set of a counter and domain separation bit (i, j). In iCOFB, a TRF is called for each message/ciphertext block. Feedback functions ρ and ρ' are used to map a pair of TRF output Y_i and plaintext/ciphertext block M_i/C_i to

[1] Compared to the XOR-only condition in [NMSS18], we also accept linear operations that are similarly lightweight.

Table 1: Lightweight criteria [NMSS18] and AEAD modes. BC indicates block cipher. The "No extra state" column shows the number of extra bits if the criterion is not satisfied.

			Lightweight criteria						
AEAD	Primitive	Security	No extra state	Inv. free	Linear only	Online	Rate	Parallel.	Ref.
GCM	BC	$O(2^{b/2})$	$4b$	✓	—	✓	< 1	✓	[MV04]
COFB	BC	$O(2^{b/2})$	$b/2$	✓	✓	✓	1	—	[CIMN17]
SAEB	BC	$O(2^{b/2})$	✓	✓	✓	✓	1/2	—	[NMSS18]
TAE	TBC	$O(2^b)$	b	—	✓	✓	1	✓	[LRW02]
ΘCB3	TBC	$O(2^b)$	b	—	✓	✓	1	✓	[KR11]
$\mathbb{OTR}$	TBC	$O(2^b)$	$2b$	✓	✓	✓	1	✓	[Min14]
SCT	TBC	$O(2^b)$	b	✓	✓	—	1/2	✓	[PS16]
ZAE	TBC	$O(2^b)$	$4b$	✓	✓	—	2/3	✓	[IMPS17]
PFB	TBC	$O(2^b)$	✓	✓	✓	✓	1	—	**Ours**

The rate of GCM is "< 1" because field multiplication for GHASH requires a negligible cost.
AD should be fed later to realize ΘCB3 with b extra bits.

the next TRF input X_{i+1} and a ciphertext/plaintext block C_i/M_i. ρ and ρ' are defined by the $2b \times 2b$ binary matrices:

$$\begin{aligned} \rho(Y_i, M_i) &= \begin{pmatrix} X_{i+1} \\ C_i \end{pmatrix} = \begin{pmatrix} E_{1,1} & E_{1,2} \\ E_{2,1} & E_{2,2} \end{pmatrix} \begin{pmatrix} Y_i \\ M_i \end{pmatrix} \enspace , \\ \rho'(Y_i, C_i) &= \begin{pmatrix} X_{i+1} \\ M_i \end{pmatrix} = \begin{pmatrix} D_{1,1} & D_{1,2} \\ D_{2,1} & D_{2,2} \end{pmatrix} \begin{pmatrix} Y_i \\ C_i \end{pmatrix} \enspace . \end{aligned} \tag{1}$$

After consuming all the message blocks, a TRF is called once again to generate a tag T. It has been proven that iCOFB has $O(2^b)$ security with ρ and ρ' satisfying a certain criterion (see Section 3.1).

As shown in Figure 1, iCOFB needs no extra state besides the ones within the underlying TRF. In addition, iCOFB takes message/ciphertext blocks online and does not need TRF inverse. Moreover, the functions ρ and ρ' can be realized with linear operations only. Therefore, iCOFB satisfies all requirements regarding TRF-based AEAD.

Three problems are faced when designing a TBC-based AEAD mode from iCOFB's idea. First, as AD is a part of a tweak, the underlying TRF should accept an arbitrary-length tweak. On the contrary, lightweight TBCs, such as SKINNY, accept only fixed-length tweaks. XT tweak extension [MI15] can solve the problem, but it is costly as it requires a universal hash function accepting an arbitrary-length data. Second, iCOFB cannot handle arbitrary-length message because the functions ρ and ρ' accept b-bit blocks only. Third, unlike conventional schemes such as ΘCB3, iCOFB has worse security bound that depends on the maximum message block length ℓ_{max}.[2] That means a short key life for a large ℓ_{max} which results in an additional cost for rekeying or a shorter product lifetime.

1.2 Contribution

We design a (fixed tweak-length) TBC-based (and nonce-based) AEAD mode, called PFB (Plaintext FeedBack), which solves the three above-mentioned problems and satisfies all the lightweight criteria presented in Table 1. Moreover, the encryption/decryption procedures of PFB are efficient, i.e., rate-1.[3]

[2] iCOFB's security bound is $O(\ell_{\mathsf{max}} q/2^b)$ wherein q is the total number of encryption queries and forgery attempts. ΘCB3's security bound is $O(q_{\mathcal{D}}/2^b)$ wherein $q_{\mathcal{D}}$ is the number of forgery attempts.

[3] Notably, regarding parallelizability, only the encryption procedure in PFB is parallelizable. In Table 1, GCM, TAE, ΘCB3, $\mathbb{OTR}$, SCT, and ZAE are parallelizable. Regarding misuse settings, we do not claim

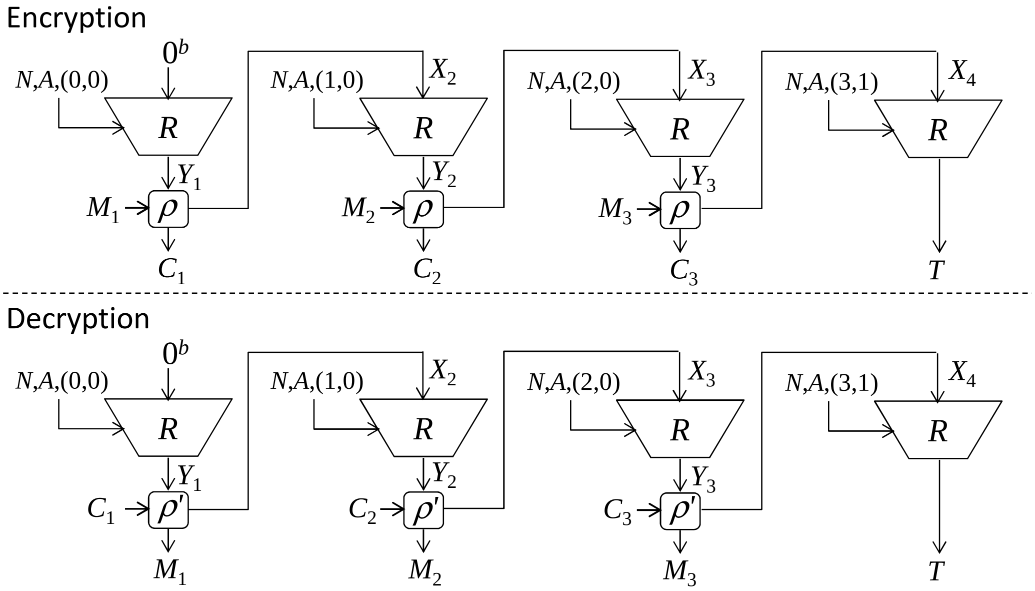

Figure 1: iCOFB. M_1, M_2, M_3 are b-bit plaintext blocks, C_1, C_2, C_3 are b-bit ciphertext blocks, and T is a b-bit tag.

We address the first problem by designing an AD processing part with a CBC-style (and Sponge-style) structure. In this part, a given AD is processed block-by-block by using a fixed-tweak-length TBC and a feedback function $\delta_*^{(a)}(W_i, A_i) = V_{i+1} = A_i \oplus W_i$, which defines the next TBC input V_{i+1} from a current AD block A_i and the previous TBC output block W_i.

We design the encryption and decryption procedures by following iCOFB's idea: plaintext/ciphertext blocks are processed by iterating a combination of TBC and a feedback function. With the aim of encrypting plaintext blocks parallelly, the PFB's functions feedback plaintext and are given by

$$\rho = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \ , \ \rho' = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \tag{2}$$

which is a member of ρ and ρ'. For a generalization in the security proof, the feedback functions are separated into the following parts:

$$\delta_*^{(e)}(Y_i, M_i) = X_{i+1} = M_i, \qquad \delta_*^{(d)}(Y_i, C_i) = X_{i+1} = Y_i \oplus C_i,$$
$$\gamma_*^{(e)}(Y_i, M_i) = C_i = Y_i \oplus M_i, \qquad \gamma_*^{(d)}(Y_i, C_i) = M_i = Y_i \oplus C_i,$$

In the encryption procedure, as a current plaintext block M_i becomes the next TBC input, the underlying TBC can be processed in parallel.[4] This feature is desirable for communication between entities with asymmetric resources, e.g., a central server sends encrypted commands to many resource-constrained nodes.

We address the second problem by incorporating one-zero padding and truncation into the abovementioned functions $\delta_*^{(a)}$, $\delta_*^{(e)}$, $\delta_*^{(d)}$, $\gamma_*^{(e)}$, and $\gamma_*^{(d)}$, which enable us to handle arbitrary-length data. The modified functions are denoted by $\delta^{(a)}$, $\delta^{(e)}$, $\delta^{(d)}$, $\gamma^{(e)}$, and

the security of PFB in the nonce-misuse setting or releasing the unverified plaintext setting. In Table 1, only SCT and ZAE have the nonce-misuse security.

[4] The AD processing part is not parallelizable and a tag is defined using the last TBC output to encrypt/decrypt the last plaintext/ciphertext block. The decryption of PFB is not parallelizable.

$\gamma^{(d)}$, and can handle data blocks of length $\leq b$ unlike the feedback functions ρ and ρ' of iCOFB.

To reduce a TBC call, padding is not performed for a full b-bit block data, and therefore, there are distinct encryption/decryption queries that become identical after one-zero padding. To avoid attacks by such data, PFB changes the corresponding tweak values. Determining a proper rule is not a trivial task: several AEAD schemes are broken by the flaws of the tweak rules [MI17] on PMACx, PMAC2x, and SIVx [LN17].

We address the third problem by providing a new security proof. By following the research direction of generalizing a cryptographic scheme without specifying a structure, such as the Stam's compression function [Sta09] as a generalization of PGV [PGV93] and Parazoa [AMP12] as a generalization of Sponge [BDPA07], we only give sufficient conditions on the generalized functions $\delta^{(a)}$, $\delta^{(e)}$, $\delta^{(d)}$, $\gamma^{(e)}$, and $\gamma^{(d)}$ regarding the inputs and outputs. The key difference in the new generalized functions is that, unlike ρ and ρ', they support data blocks with length $\leq b$. We subsequently prove that the generalized AEAD scheme, called GFB, satisfies the security bound of $O(q_{\mathcal{D}}/2^b)$ — the same level of security as ΘCB3. The generalized functions cover the PFB functions, and thus, the security bound holds for PFB.

Finally, the benefit of PFB is evaluated through concrete hardware implementations. In the implementations, PFB is instantiated with a lightweight TBC SKINNY-64-192. Its performance is compared with that of the state-of-the-art block-cipher-based alternative with the same level of security: SAEB instantiated with GIFT-128-128. For each AEAD, we evaluate the performances with and without TI. We show that PFB benefits from the small block length and shows particularly good performance in implementation with the SCA countermeasure: it has the smallest circuit area compared to SAEB implementation and the conventional implementations of Ascon [GWDE15]) and Ketje [ANR18].

1.3 Organization

The rest of this paper is organized as follows. In Section 2, we briefly review TBC and AEAD. Then, we describe the design principle and definition of GFB in Section 3, followed by its security result in Section 4. We describe the hardware implementations and their performance comparison in Section 5.

1.4 Independent Concurrent Work

Iwata et al. independently and concurrently designed Romulus [IKMP19a, IKMP19b] having several similarities to GFB, and submitted to the lightweight crypto standardization process [NIS18].[5] We summarize the key differences between the two schemes:

1. Romulus-N processes AD blocks using both tweak and input-block spaces of the underlying TBC, while GFB does not use tweak spaces.

2. The feedback function of GFB is more general than that of Romulus-N. Moreover, one instantiation of the feedback function PFB ensures that the encryption is parallelizable, while Romulus-N is not.

3. Romulus-M is secure under the nonce-misuse setting but is not online (GFB and Romulus-N do not have such misuse resistance).

[5] We submitted the preliminary version of this paper to Cryptology ePrint archive [NS19] before the NIST's deadline [NIS18]. The Romulus team follows the proof of our preliminary work [NS19] to improve Romulus's security bound to $O(q_{\mathcal{D}}/2^b)$ [IKMP19a].

2 Preliminaries

2.1 Notation

Let ε be an empty string and $\{0,1\}^*$ be the set of all bit strings. For an integer $i \geq 0$, let $\{0,1\}^i$ be the set of all i-bit strings, $\{0,1\}^0 := \{\varepsilon\}$, and $\{0,1\}^{\leq i} := \{0,1\}^1 \cup \{0,1\}^2 \cup \cdots \cup \{0,1\}^i$ be the set of all bit strings of length at most i, except for ε. Let 0^i resp. 1^i be the bit string of i-bit zeros resp. ones. For an integer $i \geq 1$, let $[i] := \{1, 2, \ldots, i\}$ be the set of positive integers less than or equal to i, and $(i) := \{0\} \cup [i]$. For a non-empty set $\mathcal{T}$, $T \xleftarrow{\$} \mathcal{T}$ means that an element is chosen uniformly at random from $\mathcal{T}$ and is assigned to T. The concatenation of two-bit strings X and Y is written as $X \| Y$ or XY when no confusion is possible. For integers $0 \leq i \leq j$ and $X \in \{0,1\}^j$, let $\mathsf{msb}_i(X)$ resp. $\mathsf{lsb}_i(X)$ be the most resp. least significant i bits of X, and $|X|$ be the number of bits of X, i.e., $|X| = j$. For integers i and j with $0 \leq i < 2^j$, let $\mathsf{str}_j(i)$ be the j-bit binary representation of i. For an integer $b \geq 0$ and a bit string X, we denote the parsing into fixed-length b-bit strings as $(X_1, X_2, \ldots, X_\ell) \xleftarrow{b} X$, where if $X \neq \varepsilon$, then $X = X_1 \| X_2 \| \cdots \| X_\ell$, $|X_i| = b$ for $i \in [\ell - 1]$, and $0 < |X_\ell| \leq b$; if $X = \varepsilon$, then $\ell = 1$ and $X = X_1 = \varepsilon$. For an integer $b > 0$, let $\mathsf{ozp}_b : \left(\{\varepsilon\} \cup \{0,1\}^{\leq b}\right) \to \{0,1\}^b$ be a one-zero padding function: for a bit string $X \in \{\varepsilon\} \cup \{0,1\}^{\leq b}$, $\mathsf{ozp}_b(X) = X$ if $|X| = b$; $\mathsf{ozp}_b(X) = X \| 10^{b-1-|X|}$ if $|X| < b$.

2.2 Tweakable Block Cipher

A tweakable block cipher (TBC) is a set of permutations indexed by a key and a public input, called tweak. Let $\mathcal{K}$ be the key space, $\mathcal{TW}$ be the tweak space, and b be the input/output-block size. A TBC (encryption) is denoted by $\widetilde{E} : \mathcal{K} \times \mathcal{TW} \times \{0,1\}^b \to \{0,1\}^b$. A TBC having a key $K \in \mathcal{K}$ is denoted by $\widetilde{E}_K$, and $\widetilde{E}_K$ having a tweak $TW \in \mathcal{TW}$ is denoted by $\widetilde{E}_K^{TW}$.

In this paper, a keyed TBC is assumed to be a secure tweakable-pseudo-random permutation (TPRP), which is indistinguishable from a tweakable random permutation (TRP). A tweakable permutation (TP) $\widetilde{P} : \mathcal{TW} \times \{0,1\}^b \to \{0,1\}^b$ is a set of b-bit permutations indexed by a tweak in $\mathcal{TW}$. A TP having a tweak $TW \in \mathcal{TW}$ is denoted by $\widetilde{P}^{TW}$. Let $\widetilde{\mathsf{Perm}}(\mathcal{TW}, \{0,1\}^b)$ be the set of all TPs with b-bit blocks and tweak space $\mathcal{TW}$. A TRP is defined as $\widetilde{P} \xleftarrow{\$} \widetilde{\mathsf{Perm}}(\mathcal{TW}, \{0,1\}^b)$. In the TPRP-security game, an adversary $\mathbf{A}$ has access to either the target keyed TBC $\widetilde{E}_K$ for $K \xleftarrow{\$} \mathcal{K}$ or a TRP $\widetilde{P} \xleftarrow{\$} \widetilde{\mathsf{Perm}}(\mathcal{TW}, \{0,1\}^b)$. After the interaction, $\mathbf{A}$ returns a decision bit $y \in \{0,1\}$. The output of $\mathbf{A}$ with access to an oracle $\mathcal{O}$ is denoted by $\mathbf{A}^{\mathcal{O}}$. For a TBC $\widetilde{E}$, the TPRP-security advantage function of an adversary $\mathbf{A}$ is defined as

$$\mathbf{Adv}^{\mathsf{tprp}}_{\widetilde{E}_K}(\mathbf{A}) := \Pr\left[K \xleftarrow{\$} \mathcal{K}; \mathbf{A}^{\widetilde{E}_K} = 1\right] - \Pr\left[\widetilde{P} \xleftarrow{\$} \widetilde{\mathsf{Perm}}(\mathcal{TW}, \{0,1\}^b); \mathbf{A}^{\widetilde{P}} = 1\right],$$

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

The maximum over all adversaries, running in time at most t and making at most σ queries, is denoted by

$$\mathbf{Adv}^{\mathsf{tprp}}_{\widetilde{E}_K}(\sigma, t) := \max_{\mathbf{A}} \mathbf{Adv}^{\mathsf{tprp}}_{\widetilde{E}_K}(\mathbf{A}) \ .$$

2.3 Nonce-Based Authenticated Encryption with Associated Data

A nonce-based authenticated encryption with associated data (nAEAD) scheme based on a keyed TBC $\widetilde{E}_K$, denoted by $\Pi[\widetilde{E}_K]$, is a pair of encryption and decryption algorithms $(\Pi.\mathsf{Enc}[\widetilde{E}_K], \Pi.\mathsf{Dec}[\widetilde{E}_K])$. $\mathcal{K}, \mathcal{N}, \mathcal{M}, \mathcal{C}, \mathcal{A}$ and $\mathcal{T}$ are the sets of keys, nonces, plaintexts, ciphertexts, associated data (AD), and tags of $\Pi[\widetilde{E}_K]$, respectively. In this paper, the key space of $\Pi[\widetilde{E}_K]$ is equal to that of the underlying TBC. The encryption algorithm takes a nonce $N \in \mathcal{N}$, AD $A \in \mathcal{A}$, and a plaintext $M \in \mathcal{M}$, and returns, deterministically, a pair of a ciphertext $C \in \mathcal{C}$ and a tag $T \in \mathcal{T}$. The decryption algorithm takes a tuple $(N, A, C, T) \in \mathcal{N} \times \mathcal{A} \times \mathcal{C} \times \mathcal{T}$, and returns, deterministically, either the distinguished invalid (reject) symbol $\bot \not\in \mathcal{M}$ or a plaintext $M \in \mathcal{M}$. We require $|\Pi.\mathsf{Enc}[\widetilde{E}_K](N, A, M)| = |\Pi.\mathsf{Enc}[\widetilde{E}_K](N, A, M')|$ when these outputs are strings and $|M| = |M'|$. We also require that $\Pi[\widetilde{E}_K]$ is correct: $\forall(K, N, A, M) \in \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} : \Pi.\mathsf{Dec}[\widetilde{E}_K](N, A, \Pi.\mathsf{Enc}[\widetilde{E}_K](N, A, M)) = M$. We consider two security notions of nAEAD, privacy and authenticity. Hereafter, we call queries to the encryption resp. decryption oracle "encryption queries" resp. "decryption queries."

Privacy

The privacy notion considers the indistinguishability between the encryption $\Pi.\mathsf{Enc}[\widetilde{E}_K]$ and a random-bit oracle \$, in the nonce-respecting setting (all nonces in encryption queries are distinct). \$ has the same interface as $\Pi.\mathsf{Enc}[\widetilde{E}_K]$, and for a query (N, A, M), returns a random bit string of length $|\Pi.\mathsf{Enc}[\widetilde{E}_K](N, A, M)|$. In the privacy game, a nonce-respecting adversary $\mathbf{A}$ interacts with either $\Pi.\mathsf{Enc}[\widetilde{E}_K]$ or \$, and then returns a decision bit $y \in \{0, 1\}$. The privacy advantage function of an adversary $\mathbf{A}$ is defined as

$$\mathbf{Adv}^{\mathsf{priv}}_{\Pi[\widetilde{E}_K]}(\mathbf{A}) := \Pr[K \xleftarrow{\$} \mathcal{K}; \mathbf{A}^{\Pi.\mathsf{Enc}[\widetilde{E}_K]} = 1] - \Pr[\mathbf{A}^{\$} = 1] \ ,$$

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

The maximum over all adversaries, running in at most t time and making encryption queries of $\sigma_{\mathcal{E}}$ the total number of TBC calls invoked by all encryption queries, is denoted by

$$\mathbf{Adv}^{\mathsf{priv}}_{\Pi[\widetilde{E}_K]}(\sigma_{\mathcal{E}}, t) := \max_{\mathbf{A}} \mathbf{Adv}^{\mathsf{priv}}_{\Pi[\widetilde{E}_K]}(\mathbf{A}) \ .$$

When an adversary is a computationally unbounded algorithm, the time t is disregarded.

Authenticity

The authenticity notion considers the unforgeability in the nonce-respecting setting. In the authenticity game, a nonce-respecting adversary $\mathbf{A}$ interacts with $\Pi[\widetilde{E}_K] = (\Pi.\mathsf{Enc}[\widetilde{E}_K], \Pi.\mathsf{Dec}[\widetilde{E}_K])$, and aims to make a non-trivial decryption query whose response is not $\bot$. The authenticity advantage of an adversary $\mathbf{A}$ is defined as

$$\mathbf{Adv}^{\mathsf{auth}}_{\Pi[\widetilde{E}_K]}(\mathbf{A}) := \Pr[K \xleftarrow{\$} \mathcal{K}; \mathbf{A}^{\Pi.\mathsf{Enc}[\widetilde{E}_K], \Pi.\mathsf{Dec}[\widetilde{E}_K]} \text{ forges}] \ ,$$

where the probabilities are taken over K and $\mathbf{A}$. We demand that $\mathbf{A}$ is nonce-respecting (all nonces in encryption queries are distinct and the condition is not for decryption queries), that $\mathbf{A}$ never asks a trivial decryption query (N, A, C, T), i.e., there is a prior encryption query (N, A, M) with $(C, T) = \Pi.\mathsf{Enc}[\widetilde{E}_K](N, A, M)$, and that $\mathbf{A}$ never repeats a query. $\mathbf{A}^{\Pi.\mathsf{Enc}[\widetilde{E}_K], \Pi.\mathsf{Dec}[\widetilde{E}_K]}$ forges means that $\mathbf{A}$ makes a decryption query whose response is not $\bot$.

The maximum over all adversaries, running in at most t time and making at most $q_{\mathcal{E}}$ encryption queries and $q_{\mathcal{D}}$ decryption queries of σ the total number of TBC calls invoked by all queries, is denoted by

$$\mathbf{Adv}^{\mathsf{auth}}_{\Pi[\widetilde{E}_K]}((q_{\mathcal{E}}, q_{\mathcal{D}}, \sigma), t) := \max_{\mathbf{A}} \mathbf{Adv}^{\mathsf{auth}}_{\Pi[\widetilde{E}_K]}(\mathbf{A}) \ .$$

When an adversary is a computationally unbounded algorithm, the time t is disregarded.

3 PFB: Lightweight TBC-based nAEAD Mode of Operation

We design a TBC-based nAEAD scheme using the iCOFB design approach.

3.1 Brief Overview of iCOFB Design and Security

As mentioned in Section 1, iCOFB [CIMN17] has the structure of iterating a combination of a TRF and the linear feedback function ρ/ρ' defined in Eq. (1). The specification of iCOFB is given in Figure 1.

To ensure the correctness of iCOFB, the feedback functions ρ/ρ' with the following conditions are used: $E_{2,2}$ is invertible; $D_{1,1} = E_{1,1} + E_{1,2}E_{2,2}^{-1}E_{2,1}$; $D_{1,2} = E_{1,2}E_{2,2}^{-1}$; $D_{2,1} = E_{2,2}^{-1}E_{2,1}$; $D_{2,2} = E_{2,2}^{-1}$.

Regarding the security of iCOFB, they show the following theorem.

Theorem 1 *[Security of iCOFB] If the feedback functions ρ/ρ' satisfy the following conditions: **(A1)** $E_{2,1}$ is invertible; **(A2)** $D_{1,2}$ is invertible; **(A3)** $D_{1,1}$ is invertible, then for any adversary $\mathbf{A}$ making at most $q_{\mathcal{D}}$ decryption queries of*

$$\mathbf{Adv}^{\mathsf{priv}}_{\mathsf{iCOFB}[R]}(\sigma_{\mathcal{E}}) = 0 \ , \quad \mathbf{Adv}^{\mathsf{auth}}_{\mathsf{iCOFB}[R]}(q_{\mathcal{E}}, q_{\mathcal{D}}, q_{\mathcal{D}}\ell_{\mathsf{max}}) \leq \frac{q_{\mathcal{D}}(\ell_{\mathsf{max}}+1)}{2^b} \ ,$$

where ℓ_{max} is the maximum query length in blocks.

3.2 Design Principle and Specification of PFB

We design PFB (stands for Plaintext FeedBack), a TBC-based lightweight nAEAD mode, by extending the idea of iCOFB with TBC. With the aim of performing the encryption parallelly, we choose the feedback functions in Eq. (2) from the class of feedback functions ρ/ρ'. However, as mentioned in Section 1, we need to solve the following three problems.

- The first problem is that iCOFB requires a TRF that accepts arbitrary-length AD.
- The second problem is that the feedback functions ρ and ρ' handle only b-bit data blocks, i.e., iCOFB handles only plaintexts and ciphertexts of length multiple of b.
- The third problem is that the security of iCOFB depends on the data length ℓ_{max}.

Hereafter, PFB is designed such that the above three problems are solved.

Hash Procedure (AD Processing)

To design a lightweight nAEAD scheme, PFB uses a fixed-tweak-length TBC, whereas iCOFB uses a variable-tweak-length TRF to take variable-length AD. Hence, we define an additional procedure of processing variable-length AD. In PFB, (arbitrary-length) AD is partitioned into b-bit blocks (the length of the last block is $\leq b$), and then, the AD blocks are processed by iterating a combination of a TBC and the following feedback function $\delta^{(a)} : \{0,1\}^b \times \left(\{\varepsilon\} \cup \{0,1\}^{\leq b}\right) \to \{0,1\}^b$:

$$V_i \leftarrow \delta^{(a)}(W_{i-1}, A_i) := W_{i-1} \oplus \mathsf{ozp}_b(A_i),$$

where $A_i \in \{\varepsilon\} \cup \{0,1\}^{\leq b}$ is a current AD block, $W_{i-1} \in \{0,1\}^b$ is the previous TBC output, and $V_i \in \{0,1\}^b$ is the next TBC input. The core procedure of the hash of PFB is given on the left of Figure 2. This procedure can handle arbitrary-length AD, and thus, the first problem is solved. Note that when AD is an empty string, the above procedure is performed for $A_1 = \varepsilon$.

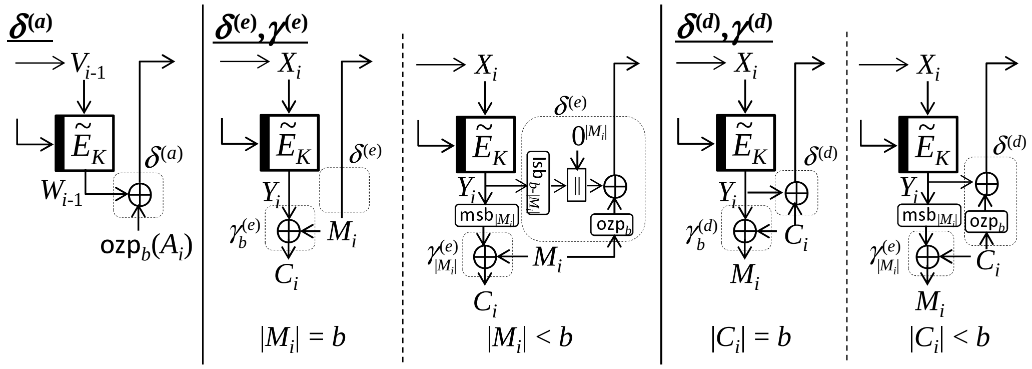

Figure 2: Core procedures of PFB: $\delta^{(a)}, \delta^{(e)}, \delta^{(d)}, \gamma^{(e)}, \gamma^{(d)}$. A_i is an i-th AD block. M_i is an i-th plaintext block. C_i is an i-th ciphertext block. Tweaks are omitted.

Encryption/Decryption Procedures

In PFB, a plaintext/ciphertext is partitioned into b-bit blocks, and as iCOFB, each block is processed by a TBC and a feedback function.

- The feedback function in the encryption is composed of the following two functions, where $0 < l \leq b$ is an integer, $M_i \in \{0,1\}^l$ is a current plaintext block, $Y_i \in \{0,1\}^b$ is the previous TBC output, $X_{i+1} \in \{0,1\}^b$ is the next TBC input, and $C_i \in \{0,1\}^l$ is the ciphertext block.

  – $\gamma_l^{(e)} : \{0,1\}^l \times \{0,1\}^l \to \{0,1\}^l$ is defined as

  $$C_i \leftarrow \gamma_l^{(e)}(\mathsf{msb}_l(Y_i), M_i) := \mathsf{msb}_l(Y_i) \oplus M_i.$$

  – $\delta^{(e)} : \{0,1\}^b \times \{0,1\}^{\leq b} \to \{0,1\}^b$ is defined as

  $$X_{i+1} \leftarrow \delta^{(e)}(Y_i, M_i) := \mathsf{ozp}_b(M_i) \oplus \left(0^{|M_i|} \| \mathsf{lsb}_{b-|M_i|}(Y_i)\right).$$

  The core procedure of the encryption of PFB is given in the center of Figure 2. Note that plaintext blocks, except for the last block, are of $l = b$, and the last block is of $l \leq b$.

- The feedback function in the decryption is composed of the following two functions, where $0 < l \leq b$ is an integer, $C_i \in \{0,1\}^l$ is a current ciphertext block, $Y_i \in \{0,1\}^l$ is the previous TBC output, $X_{i+1} \in \{0,1\}^b$ is the next TBC input, and $M_i \in \{0,1\}^l$ is the plaintext block.

  – $\gamma_l^{(d)} : \{0,1\}^l \times \{0,1\}^l \to \{0,1\}^l$ is defined as

  $$M_i \leftarrow \gamma_l^{(d)}(\mathsf{msb}_l(Y_i), C_i) := \mathsf{msb}_l(Y_i) \oplus C_i.$$

  – $\delta^{(d)} : \{0,1\}^b \times \{0,1\}^{\leq b} \to \{0,1\}^b$ is defined as

  $$X_{i+1} \leftarrow \delta^{(d)}(Y_i, C_i) := Y_i \oplus \mathsf{ozp}_b(C_i).$$

  The core procedure of the decryption of PFB is given on the right of Figure 2. Note that ciphertext blocks, except for the last block, are of $l = b$, and the last block is of $l \leq b$.

The above functions enable us to handle arbitrary-length plaintexts/ciphertexts, and thus, the second problem is solved.

Tweak Function

Let ℓ_{max} be the maximum block size of AD, plaintext, and ciphertext. In PFB, the following tweak function is used:

- $f : [7] \times \mathcal{N} \times (\ell_{\mathsf{max}}) \to \mathcal{TW}$,

with the following condition:

- **B1**: for any $(i, N, j), (i', N', j') \in [7] \times \mathcal{N} \times (\ell_{\mathsf{max}})$ such that $(i, N, j) \neq (i', N', j')$,
$$f(i, N, j) \neq f(i', N', j').$$

Here, we assume that $\mathcal{N}$ includes the constant 0^n for an integer $n > 0$. The first element is used for distinguishing AD and plaintext/ciphertext, and whether the last block is a full-bit one or not, which offers a distinct permutation between the hash procedure and the encryption/decryption and which avoids a redundant TBC call when the last block is a full-bit one. The second element is a nonce, which offers a distinct permutation for each encryption (under the nonce-respecting setting), thereby removing the birthday term regarding the number of queries. The third element is the current block number, which offers a distinct permutation for each block, thereby removing the query length from the security bound.

In our hardware implementation given in Section 5, for positive integers t and n such that $n+3+1 \leq t$, we define $\mathcal{TW} := \{0,1\}^t$ and $\mathcal{N} := \{0,1\}^n$, and use the following tweak function:

$$f(i, N, j) = (\mathsf{str}_3(i) \| N \| \mathsf{str}_{t-3-n}(j)). \tag{3}$$

The function satisfies the condition **B1**.

Specification of PFB

The specification of PFB is given in Algorithm 1 and is shown in Figure 3, where $\gamma_l^{(e)}$, $\gamma_l^{(d)}$, $\delta^{(a)}$, $\delta^{(e)}$, and $\delta^{(d)}$ are defined above; PFB.Hash is the hash procedure; PFB.Enc is the encryption; and PFB.Dec is the decryption. The full specification, including the structures of $\gamma_l^{(e)}$, $\gamma_l^{(d)}$, $\delta^{(a)}$, $\delta^{(e)}$, and $\delta^{(d)}$, is given in Algorithm 2 and Figure 6 in Appendix B.

4 Security of PFB

In this section, we provide the privacy and authenticity bounds of PFB and the security proofs, which solve the third problem mentioned in Section 3.2. In our proof, with the aim of covering a broader class of feedback functions, we generalize the functions $\delta^{(a)}$, $\delta^{(e)}$, $\delta^{(d)}$, $\gamma^{(e)}$, and $\gamma^{(d)}$ that require only sufficient conditions regarding inputs and outputs of the functions.

4.1 GFB: Generalized PFB

The specification of GFB, a generalization of PFB, is given in Algorithm 3, where the structures of the functions $\delta^{(a)}$, $\delta^{(e)}$, $\delta^{(d)}$, $\gamma^{(e)}$, and $\gamma^{(d)}$ are removed and the following conditions regarding inputs and outputs are set. Considering GFB, the hash, encryption, and decryption procedures are renamed as GFB.Hash, GFB.Enc, and GFB.Dec, respectively.

For the correctness of GFB, the following conditions are required.

- **B2**: for any non-negative integer $l \leq b$, $Y \in \{0,1\}^l$, $\gamma_l^{(e)}(Y, \cdot)$ is bijective and $\gamma_l^{(d)}(Y, \cdot)$ is the inverse of $\gamma_l^{(e)}(Y, \cdot)$, i.e., $\forall M \in \{0,1\}^l : M = \gamma_l^{(d)}(Y, \gamma_l^{(e)}(Y, M))$.

Algorithm 1 PFB

```
Encryption $\mathsf{PFB.Enc}[\widetilde{E}_K](N, A, M)$
 1: $X_1 \leftarrow \mathsf{PFB.Hash}[\widetilde{E}_K](A)$
 2: if $A \neq \varepsilon \wedge |A| \mod b = 0$ then $x \leftarrow 2$; else $x \leftarrow 3$
 3: if $M = \varepsilon$ then $C \leftarrow \varepsilon$; $\ell \leftarrow 0$; goto step 8
 4: $M_1, \ldots, M_\ell \xleftarrow{b} M$
 5: for $i = 1, \ldots, \ell$ do
 6:     $Y_i \leftarrow \widetilde{E}_K^{f(x,N,i)}(X_i)$; $C_i \leftarrow \gamma^{(e)}_{|M_i|}(\mathsf{msb}_{|M_i|}(Y_i), M_i)$; $X_{i+1} \leftarrow \delta^{(e)}(Y_i, M_i)$
 7: end for
 8: if $M \neq \varepsilon \wedge |M| \mod b = 0$ then $y \leftarrow x + 2$; else $y \leftarrow x + 4$
 9: $S \leftarrow X_{\ell+1}$; $T \leftarrow \mathsf{msb}_\tau\left(\widetilde{E}_K^{f(y,N,\ell)}(S)\right)$; $C \leftarrow C_1 \| \cdots \| C_\ell$
10: return $(C, T)$

Decryption $\mathsf{PFB.Dec}[\widetilde{E}_K](N, A, C, T)$
 1: $X_1 \leftarrow \mathsf{PFB.Hash}[\widetilde{E}_K](A)$
 2: if $A \neq \varepsilon \wedge |A| \mod b = 0$ then $x \leftarrow 2$; else $x \leftarrow 3$
 3: if $C = \varepsilon$ then $M \leftarrow \varepsilon$; $\ell \leftarrow 0$; goto step 8
 4: $C_1, \ldots, C_\ell \xleftarrow{b} C$
 5: for $i = 1, \ldots, \ell$ do
 6:     $Y_i \leftarrow \widetilde{E}_K^{f(x,N,i)}(X_i)$; $M_i \leftarrow \gamma^{(d)}_{|C_i|}(\mathsf{msb}_{|C_i|}(Y_i), C_i)$; $X_{i+1} \leftarrow \delta^{(d)}(Y_i, C_i)$
 7: end for
 8: if $C \neq \varepsilon \wedge |C| \mod b = 0$ then $y \leftarrow x + 2$; else $y \leftarrow x + 4$
 9: $S \leftarrow X_{\ell+1}$; $\hat{T} \leftarrow \mathsf{msb}_\tau\left(\widetilde{E}_K^{f(y,N,\ell)}(S)\right)$; $M \leftarrow M_1 \| \cdots \| M_\ell$
10: if $T = \hat{T}$ then return $M$; else return $\perp$

Hash $\mathsf{PFB.Hash}[\widetilde{E}_K](A)$
 1: $A_1, \ldots, A_a \xleftarrow{b} A$; $W_0 \leftarrow 0^b$
 2: for $i = 1, \ldots, a - 1$ do $V_i \leftarrow \delta^{(a)}(W_{i-1}, A_i)$; $W_i \leftarrow \widetilde{E}_K^{f(1,0^n,i)}(V_i)$
 3: $V_a \leftarrow \delta^{(a)}(W_{a-1}, A_a)$; $H \leftarrow V_a$
 4: return $H$
```

- **B3**: $M \in \{0,1\}^{\leq b}, Y \in \{0,1\}^b$, $\delta^{(e)}(Y, M) = \delta^{(d)}(Y, \gamma^{(e)}_{|M|}(\mathsf{msb}_{|M|}(Y), M))$.

For GFB to be secure, we require the following five conditions on $\gamma^{(e)}_l, \gamma^{(d)}_l, \delta^{(a)}, \delta^{(e)}, \delta^{(d)}$.

- **B4**: for any $M \in \{0,1\}^{\leq b}$, $\gamma^{(e)}_{|M|}(\cdot, M)$ is bijective.
- **B5**: for any $C \in \{0,1\}^{\leq b}$, $\delta^{(d)}(\cdot, C)$ is bijective.
- **B6**: for any $C, C' \in \{0,1\}^{\leq b}$ and $Y, Y' \in \{0,1\}^b$,

$$\begin{aligned}\delta^{(e)}(Y, \gamma^{(d)}_{|C|}(\mathsf{msb}_{|C|}(Y), C)) = \delta^{(d)}(Y', C')&\\ \Rightarrow &(C = C' \wedge Y = Y') \vee (C \neq C' \wedge Y \neq Y')\\ &\vee (C \neq C' \wedge Y = Y' \wedge |C| = b \wedge |C'| < b)\\ &\vee (C \neq C' \wedge Y = Y' \wedge |C| < b \wedge |C'| = b).\end{aligned}$$

- **B7**: for any $A \in \{0,1\}^{\leq b}$, $\delta^{(a)}(\cdot, A)$ is bijective.

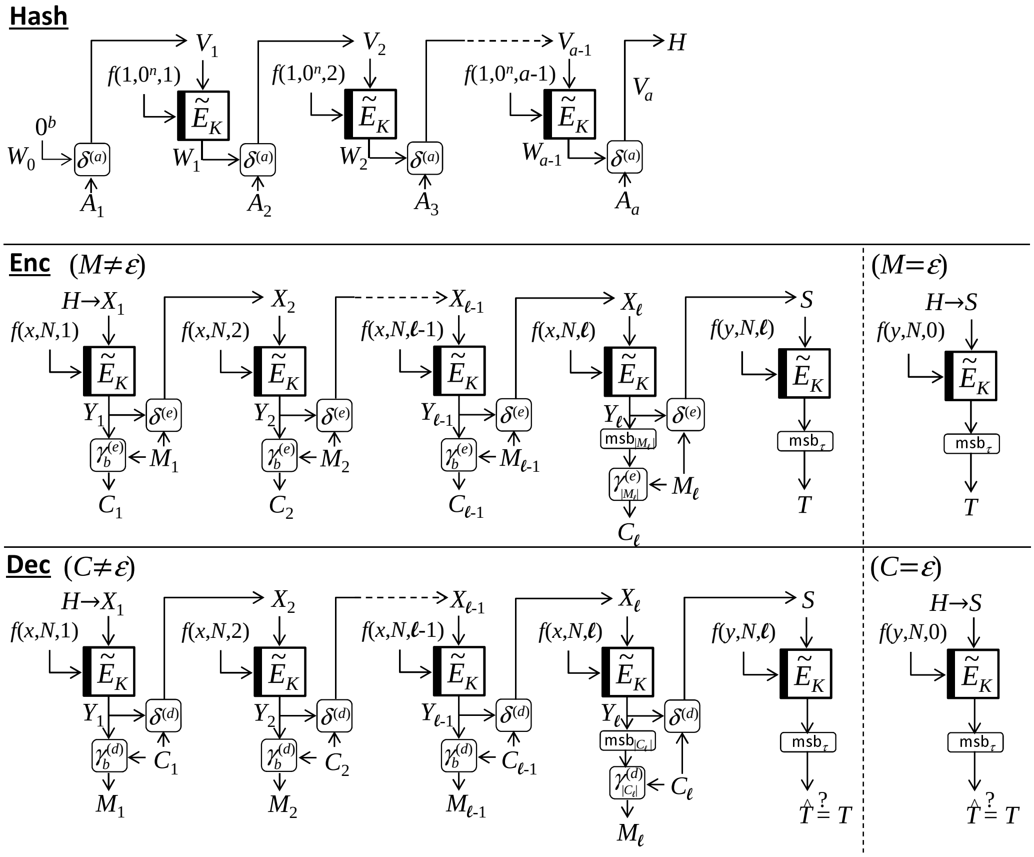

Figure 3: PFB/GFB. $A_1, \ldots, A_a \xleftarrow{b} A$. $M_1, \ldots, M_\ell \xleftarrow{b} M$ (in the encryption algorithm) and $C_1, \ldots, C_\ell \xleftarrow{b} C$ (in the decryption algorithm). If $A \neq \varepsilon \wedge |A| \mod b = 0$ then $x \leftarrow 2$; else $x \leftarrow 3$. If $C \neq \varepsilon \wedge |C| \mod b = 0$ then $y \leftarrow x+2$; else $y \leftarrow x+4$.

- **B8**: for any $A, A' \in \{\varepsilon\} \cup \{0,1\}^{\leq b}, W, W' \in \{0,1\}^b$,

$$\begin{aligned}\delta^{(a)}(W,A) = \delta^{(a)}(W,A') \Rightarrow & (A = A' \wedge W = W') \vee (A \neq A' \wedge W \neq W') \vee \\ & (A \neq A' \wedge W = W' \wedge |A| = b \wedge |A'| < b).\end{aligned}$$

The condition **B4** ensures that for a plaintext block M_i and a TBC output Y_i, if Y_i is uniformly distributed over $\{0,1\}^b$, then so is the ciphertext block $C_i = \gamma^{(e)}_{|M_i|}(Y_i, M_i)$. The condition **B5** ensures that the internal state collision $\delta^{(e)}(Y_i, M_i) = \delta^{(d)}(Y_i', C_i')$ (between the encryption and decryption) depends on the randomness of the TBC output Y_i'. Thus, if the output is distributed over a set $\mathcal{X}$, then the collision probability can be at most $1/|\mathcal{X}|$. Similarly, the condition **B7** ensures that in the procedure of processing AD blocks, the internal state collision $\delta^{(a)}(W_i, A_i) = \delta^{(a)}(W_i', A_i')$ depends on the randomness of the TBC output W_i'. The conditions **B5**, **B7** are used to upper bound the authenticity advantage, as the internal-state collision offers a forgery attack. The condition **B6** ensures that in the encryption and decryption procedures, no trivial collision occurs on the internal state values. Note that the conditions $(C \neq C' \wedge |C| = b \wedge |C'| < b \wedge Y = Y')$ and $(C \neq C' \wedge |C| < b \wedge |C'| = b \wedge Y = Y')$ in **B6** tolerate (possibly trivial) internal-state collisions but the first element of f eliminates the influence of the collisions. The condition **B8** is defined similarly.

4.2 Security Bounds and Proofs

The privacy and authenticity bounds of GFB are given in the following theorem.

Theorem 2 *[Security of* GFB*] For* GFB *with the conditions* ***B1-B8****, we have*

$$\mathbf{Adv}^{\mathsf{priv}}_{\mathsf{GFB}[\widetilde{E}_K]}(\sigma_{\mathcal{E}}, t) \le \mathbf{Adv}^{\mathsf{tprp}}_{\widetilde{E}_K}(\sigma_{\mathcal{E}}, t + O(\sigma_{\mathcal{E}})) \ ,$$
$$\mathbf{Adv}^{\mathsf{auth}}_{\mathsf{GFB}[\widetilde{E}_K]}((q_{\mathcal{E}}, q_{\mathcal{D}}, \sigma), t) \le \frac{q_{\mathcal{D}}}{2^{\tau} - 1/2^{b-\tau}} + \frac{q_{\mathcal{D}}}{2^b - 1} + \mathbf{Adv}^{\mathsf{tprp}}_{\widetilde{E}_K}(\sigma, t + O(\sigma)) \ .$$

As GFB includes PFB (see also Appendix A), the above theorem offers the following corollary.

Corollary 1 *[Security of* PFB*]*

$$\mathbf{Adv}^{\mathsf{priv}}_{\mathsf{PFB}[\widetilde{E}_K]}(\sigma_{\mathcal{E}}, t) \le \mathbf{Adv}^{\mathsf{tprp}}_{\widetilde{E}_K}(\sigma_{\mathcal{E}}, t + O(\sigma_{\mathcal{E}})) \ ,$$
$$\mathbf{Adv}^{\mathsf{auth}}_{\mathsf{PFB}[\widetilde{E}_K]}((q_{\mathcal{E}}, q_{\mathcal{D}}, \sigma), t) \le \frac{q_{\mathcal{D}}}{2^{\tau} - 1/2^{b-\tau}} + \frac{q_{\mathcal{D}}}{2^b - 1} + \mathbf{Adv}^{\mathsf{tprp}}_{\widetilde{E}_K}(\sigma, t + O(\sigma)) \ .$$

The proof of Theorem 2 is given below.

4.3 Replacing the Keyed TBC $\boldsymbol{E_K}$ with a TRP $\boldsymbol{\widetilde{P}}$

The keyed TBC $\widetilde{E}_K$ for $K \xleftarrow{\$} \mathcal{K}$ is replaced with a TRP $\widetilde{P} \xleftarrow{\$} \widetilde{\mathsf{Perm}}\left(\mathcal{TW}, \{0,1\}^b\right)$. By the replacement, we have

$$\mathbf{Adv}^{\mathsf{priv}}_{\mathsf{GFB}[\widetilde{E}_K]}(\sigma_{\mathcal{E}}, t) \le \mathbf{Adv}^{\mathsf{priv}}_{\mathsf{GFB}[\widetilde{P}]}(\sigma_{\mathcal{E}}) + \mathbf{Adv}^{\mathsf{tprp}}_{\widetilde{E}_K}(\sigma_{\mathcal{E}}, t + O(\sigma_{\mathcal{E}})) \ , \tag{4}$$
$$\mathbf{Adv}^{\mathsf{auth}}_{\mathsf{GFB}[\widetilde{E}_K]}((q_{\mathcal{E}}, q_{\mathcal{D}}, \sigma), t) \le \mathbf{Adv}^{\mathsf{auth}}_{\mathsf{GFB}[\widetilde{P}]}(q_{\mathcal{E}}, q_{\mathcal{D}}, \sigma) + \mathbf{Adv}^{\mathsf{tprp}}_{\widetilde{E}_K}(\sigma, t + O(\sigma)) \ . \tag{5}$$

Hereafter, the privacy and authenticity advantages of $\mathsf{GFB}[\widetilde{P}]$ are upper-bounded in Sections 4.4 and 4.5 (the upper bounds are given in Eqs. (6) and (7)), respectively, where adversaries are computationally unbounded algorithms and the complexities are solely measured by the numbers of queries. Without loss of generality, adversaries are deterministic.

4.4 Upper Bounding $\mathbf{Adv}^{\mathsf{priv}}_{\mathsf{GFB}[\widetilde{P}]}(\boldsymbol{\sigma_{\mathcal{E}}})$

The condition **B1** of the tweak function f ensures that tweaks of $\widetilde{P}$ (producing ciphertexts and tags) in $\mathsf{GFB.Enc}$, which are defined by encryption queries, are all distinct. Hence, the output blocks of $\widetilde{P}$ are chosen independently and uniformly at random from $\{0,1\}^b$. Note that tweaks of $\widetilde{P}$ (processing AD blocks) in $\mathsf{GFB.Hash}$ might be repeated, but the repeated tweaks do not affect the randomness of the output blocks due to the distinct tweaks in $\mathsf{GFB.Enc}$. Under condition **B4**, all ciphertext blocks C_i defined by encryption queries are independently and uniformly distributed over $\{0,1\}^{|C_i|}$ and thus are indistinguishable from those defined by \$. Hence, we have

$$\mathbf{Adv}^{\mathsf{priv}}_{\mathsf{GFB}[\widetilde{P}]}(\sigma_{\mathcal{E}}) = 0 \ . \tag{6}$$

4.5 Upper Bounding $\mathbf{Adv}^{\mathsf{auth}}_{\mathsf{GFB}[\widetilde{P}]}(\boldsymbol{q_{\mathcal{E}}, q_{\mathcal{D}}, \sigma})$

Without loss of generality, assume that an adversary $\mathbf{A}$ aborts after $\mathbf{A}$ forges. Let forge_i be an event that at the i-th decryption query, $\mathbf{A}$ forges (thus forge_i occurs as long as

$\mathsf{forge}_1 \vee \mathsf{forge}_2 \vee \cdots \vee \mathsf{forge}_{i-1}$ does not occur). We then have

$$\mathbf{Adv}^{\mathsf{auth}}_{\mathsf{GFB}[\widetilde{P}]}(q_{\mathcal{E}}, q_{\mathcal{D}}, \sigma) \leq \sum_{i=1}^{q_{\mathcal{D}}} \Pr[\mathsf{forge}_i] \ .$$

Next, $\Pr[\mathsf{forge}_i]$ is upper-bounded, where $i \in [q_{\mathcal{D}}]$. Values/variables corresponding to the i-th decryption query, except for the lengths a and ℓ, are denoted using the superscript of (d). The lengths a and ℓ are denoted by a_d and ℓ_d, respectively. Similarly, for an encryption query $(N^{(e)}, A^{(e)}, M^{(e)})$, values/variables corresponding to the encryption query, except for the lengths a and ℓ, are denoted using the superscript of (e). The lengths a and ℓ are denoted by a_e and ℓ_e, respectively. In this analysis, we consider the following types of decryption queries.

- Type-1: For any previous encryption query $(N^{(e)}, A^{(e)}, M^{(e)})$,
$$N^{(e)} \neq N^{(d)} \vee y^{(e)} \neq y^{(d)} \vee \ell_e \neq \ell_d.$$
- Type-2: For some previous encryption query $(N^{(e)}, A^{(e)}, M^{(e)})$,
$$N^{(e)} = N^{(d)} \wedge y^{(e)} = y^{(d)} \wedge \ell_e = \ell_d.$$

Then,

$$\begin{aligned}\Pr[\mathsf{forge}_i] &= \Pr[\mathsf{forge}_i \wedge \mathsf{Type\text{-}1}] + \Pr[\mathsf{forge}_i \wedge \mathsf{Type\text{-}2}] \\ &= \Pr\left[\mathsf{forge}_i | \mathsf{Type\text{-}1}\right] \cdot \Pr\left[\mathsf{Type\text{-}1}\right] + \Pr\left[\mathsf{forge}_i | \mathsf{Type\text{-}2}\right] \cdot \Pr\left[\mathsf{Type\text{-}2}\right] \\ &\leq \max\left\{\Pr\left[\mathsf{forge}_i | \mathsf{Type\text{-}1}\right], \Pr\left[\mathsf{forge}_i | \mathsf{Type\text{-}2}\right]\right\} \ .\end{aligned}$$

In Section 4.6, $\Pr\left[\mathsf{forge}_i | \mathsf{Type\text{-}1}\right]$ is analyzed, and in Section 4.7, $\Pr\left[\mathsf{forge}_i | \mathsf{Type\text{-}2}\right]$ is analyzed. The upper bounds (8), (9) give

$$\Pr[\mathsf{forge}_i] \leq \frac{1}{2^\tau - 1/2^{b-\tau}} + \frac{1}{2^b - 1} \ .$$

Thus, we have

$$\mathbf{Adv}^{\mathsf{auth}}_{\mathsf{GFB}[\widetilde{P}]}(q_{\mathcal{E}}, q_{\mathcal{D}}, \sigma) \leq q_{\mathcal{D}} \cdot \left(\frac{1}{2^\tau - 1/2^{b-\tau}} + \frac{1}{2^b - 1}\right) = \frac{q_{\mathcal{D}}}{2^\tau - 1/2^{b-\tau}} + \frac{q_{\mathcal{D}}}{2^b - 1} \ . \tag{7}$$

4.6 Analysis of $\Pr\left[\mathsf{forge}_i | \mathsf{Type\text{-}1}\right]$

Under the Type-1 decryption query and the condition **B1**, the tweak $f(y^{(d)}, N^{(d)}, \ell_d)$, with which the TRP defines the tag $\hat{T}^{(d)}$, is distinct from all tweaks defined by the previous encryption queries, as well as that from other tweaks defined by the i-th decryption query. Hence, $\hat{T}^{(d)}$ is uniformly distributed over $\{0,1\}^\tau$ and independent of the TRP outputs defined by the previous encryption queries and of other TRP outputs defined by the decryption query. Thus, we have

$$\Pr\left[\mathsf{forge}_i | \mathsf{Type\text{-}1}\right] \leq \frac{1}{2^\tau} \ . \tag{8}$$

4.7 Analysis of $\Pr[\mathsf{forge}_i|\mathsf{Type\text{-}2}]$

$\Pr\left[\hat{T}^{(d)} = T^{(d)} \middle| S^{(d)} \neq S^{(e)} \wedge \mathsf{Type\text{-}2}\right]$ and $\Pr\left[S^{(d)} = S^{(e)} \middle| \mathsf{Type\text{-}2}\right]$ are upper-bounded, because

$$\begin{aligned}\Pr\left[\mathsf{forge}_i|\mathsf{Type\text{-}2}\right] &= \Pr\left[\hat{T}^{(d)} = T^{(d)} \wedge S^{(d)} \neq S^{(e)} \middle| \mathsf{Type\text{-}2}\right] \\ &\quad + \Pr\left[\hat{T}^{(d)} = T^{(d)} \wedge S^{(d)} = S^{(e)} \middle| \mathsf{Type\text{-}2}\right] \\ &= \Pr\left[\hat{T}^{(d)} = T^{(d)} \middle| \mathsf{Type\text{-}2} \wedge S^{(d)} \neq S^{(e)}\right] \cdot \Pr\left[S^{(d)} \neq S^{(e)} \middle| \mathsf{Type\text{-}2}\right] \\ &\quad + \Pr\left[\hat{T}^{(d)} = T^{(d)} \middle| \mathsf{Type\text{-}2} \wedge S^{(d)} = S^{(e)}\right] \cdot \Pr\left[S^{(d)} = S^{(e)} \middle| \mathsf{Type\text{-}2}\right] \\ &\leq \Pr\left[\hat{T}^{(d)} = T^{(d)} \middle| \mathsf{Type\text{-}2} \wedge S^{(d)} \neq S^{(e)}\right] + \Pr\left[S^{(d)} = S^{(e)} \middle| \mathsf{Type\text{-}2}\right] \quad .\end{aligned}$$

The upper bounds (10), (13) give

$$\Pr\left[\mathsf{forge}_i|\mathsf{Type\text{-}2}\right] \leq \frac{1}{2^\tau - 1/2^{b-\tau}} + \frac{1}{2^b - 1} \quad . \tag{9}$$

Upper Bounding $\Pr\left[\hat{T}^{(d)} = T^{(d)} \middle| \mathsf{Type\text{-}2} \wedge S^{(d)} \neq S^{(e)}\right]$

For the Type-2 decryption query, by $S^{(d)} \neq S^{(e)}$ and $f(y^{(e)}, N^{(e)}, \ell_e) = f(y^{(d)}, N^{(d)}, \ell_d)$ (the tweaks are the same), the output of the last TRP call by the decryption query is chosen uniformly at random from $\{0,1\}^b \backslash \{\widetilde{P}^{f(y^{(e)}, N^{(e)}, \ell_e)}(S^{(e)})\}$. We thus have

$$\Pr\left[\hat{T}^{(d)} = T^{(d)} \middle| \mathsf{Type\text{-}2} \wedge S^{(d)} \neq S^{(e)}\right] \leq \frac{2^{b-\tau}}{2^b - 1} = \frac{1}{2^\tau - 1/2^{b-\tau}} \quad . \tag{10}$$

Upper Bounding $\Pr\left[S^{(d)} = S^{(e)} \middle| \mathsf{Type\text{-}2}\right]$

The conditions on Type-2 $y^{(e)} = y^{(d)} \wedge \ell_e = \ell_d$, are satisfied if and only if

$$\left(\left|A^{(d)}_{a_d}\right| = \left|A^{(e)}_{a_e}\right| = b\right) \vee \left(\left|A^{(d)}_{a_d}\right| < b \wedge \left|A^{(e)}_{a_e}\right| < b\right) \text{ and} \tag{11}$$

$$\left(\left|M^{(d)}_{\ell_d}\right| = \left|M^{(e)}_{\ell_e}\right| = b\right) \vee \left(0 < \left|M^{(d)}_{\ell_d}\right| < b \wedge 0 < \left|M^{(e)}_{\ell_e}\right| < b\right) \vee \left(M^{(d)} = M^{(e)} = \varepsilon\right). \tag{12}$$

Here, if $M = \varepsilon$, then $\ell = 0$, $M_\ell = \varepsilon$, and $|M_\ell| = 0$; if $A = \varepsilon$, then $a = 1$, $A_a = \varepsilon$ and $|A_a| = 0$. Let

$$I(A^{(d)}, A^{(e)}) = \left\{i \in [a_d] \middle| A^{(d)}_i \neq A^{(e)}_i\right\} \text{ and } I(C^{(d)}, C^{(e)}) = \left\{i \in [\ell_d] \middle| C^{(d)}_i \neq C^{(e)}_i\right\}$$

be sets of distinct blocks obtained from $(A^{(d)}, A^{(e)})$ and $(C^{(d)}, C^{(e)})$, respectively, where for $a < i$, $A_i := \varepsilon$. For convenience, we define $I(C^{(d)}, C^{(e)}) = 0$ if $C^{(d)} = C^{(e)} = \varepsilon$. Using the observation and notations, $\Pr\left[S^{(d)} = S^{(e)} \middle| \mathsf{Type\text{-}2}\right]$ is upper-bounded.

Before providing a detailed analysis, we show the sub-cases used in the analysis.

- In the first case, the ciphertexts are the same, i.e., $|I(C^{(d)}, C^{(e)})| = 0$, and the AD lengths are the same, i.e., $a_e = a_d$ (note that $A_e \neq A_d$). The analysis of this case corresponds to that of $p_{1,1}$.
- In the second case, the ciphertexts are the same, i.e., $|I(C^{(d)}, C^{(e)})| = 0$, while the AD lengths are distinct, i.e., $a_e \neq a_d$ (note that $A_e \neq A_d$). The analysis of this case corresponds to that of $p_{1,2}$.

- In the third case, the ciphertexts are distinct, i.e., $|I(C^{(d)}, C^{(e)})| \geq 1$. The analysis of this case corresponds to that of p_2.

A detailed analysis is given below.

$$\begin{aligned}
&\Pr\left[S^{(d)} = S^{(e)} \middle| \mathsf{Type\text{-}2}\right] \\
&= \underbrace{\Pr\left[S^{(d)} = S^{(e)} \wedge |I(C^{(d)}, C^{(e)})| = 0 \middle| \mathsf{Type\text{-}2}\right]}_{=:p_1} \\
&\qquad + \Pr\left[S^{(d)} = S^{(e)} \wedge |I(C^{(d)}, C^{(e)})| \geq 1 \middle| \mathsf{Type\text{-}2}\right] \\
&= p_1 + \underbrace{\Pr\left[S^{(d)} = S^{(e)} \middle| \mathsf{Type\text{-}2} \wedge |I(C^{(d)}, C^{(e)})| \geq 1\right]}_{=:p_2} \cdot \Pr\left[|I(C^{(d)}, C^{(e)})| \geq 1 \middle| \mathsf{Type\text{-}2}\right] .
\end{aligned}$$

Regarding p_1, under the condition **B6**, for $Y, Y' \in \{0,1\}^b$ and $C \in \{0,1\}^{\leq b}$,

$$\delta^{(e)}(Y, \gamma^{(d)}_{|C|}(\mathsf{msb}_{|C|}(Y), C)) = \delta^{(d)}(Y', C) \Rightarrow Y = Y'.$$

Hence, by $|I(C^{(d)}, C^{(e)})| = 0$, $S^{(d)} = S^{(e)} \Rightarrow H^{(d)} = H^{(e)}$ is satisfied, and we thus have

$$\begin{aligned}
p_1 &= \Pr\left[H^{(d)} = H^{(e)} \wedge |I(C^{(d)}, C^{(e)})| = 0 \middle| \mathsf{Type\text{-}2}\right] \\
&= \Pr\left[H^{(d)} = H^{(e)} \wedge a_e = a_d \wedge |I(C^{(d)}, C^{(e)})| = 0 \middle| \mathsf{Type\text{-}2}\right] \\
&\quad + \Pr\left[H^{(d)} = H^{(e)} \wedge a_e \neq a_d \wedge |I(C^{(d)}, C^{(e)})| = 0 \middle| \mathsf{Type\text{-}2}\right] \\
&= \underbrace{\Pr\left[H^{(d)} = H^{(e)} \middle| \mathsf{Type\text{-}2} \wedge a_e = a_d \wedge |I(C^{(d)}, C^{(e)})| = 0\right]}_{=:p_{1,1}} \\
&\quad \cdot \Pr\left[a_e = a_d \middle| \mathsf{Type\text{-}2} \wedge |I(C^{(d)}, C^{(e)})| = 0\right] \cdot \Pr\left[|I(C^{(d)}, C^{(e)})| = 0 \middle| \mathsf{Type\text{-}2}\right] \\
&\quad + \underbrace{\Pr\left[H^{(d)} = H^{(e)} \middle| \mathsf{Type\text{-}2} \wedge a_e \neq a_d \wedge |I(C^{(d)}, C^{(e)})| = 0\right]}_{=:p_{1,2}} \\
&\quad \cdot \Pr\left[a_e \neq a_d \middle| \mathsf{Type\text{-}2} \wedge |I(C^{(d)}, C^{(e)})| = 0\right] \cdot \Pr\left[|I(C^{(d)}, C^{(e)})| = 0 \middle| \mathsf{Type\text{-}2}\right] \\
&\leq \max\left\{p_{1,1}, p_{1,2}\right\} \cdot \Pr\left[|I(C^{(d)}, C^{(e)})| = 0 \middle| \mathsf{Type\text{-}2}\right] .
\end{aligned}$$

Using these upper bounds, we have

$$\begin{aligned}
\Pr\left[S^{(d)} = S^{(e)} \middle| \mathsf{Type\text{-}2}\right] &\leq \max\left\{p_{1,1}, p_{1,2}\right\} \cdot \Pr\left[|I(C^{(d)}, C^{(e)})| = 0 \middle| \mathsf{Type\text{-}2}\right] \\
&\qquad + p_2 \cdot \Pr\left[|I(C^{(d)}, C^{(e)})| \geq 1 \middle| \mathsf{Type\text{-}2}\right] \\
&\leq \max\left\{p_{1,1}, p_{1,2}, p_2\right\} .
\end{aligned}$$

$p_{1,1}, p_{1,2}, p_2$ are upper bounded below.

- $p_{1,1} = \Pr\left[H^{(d)} = H^{(e)} \middle| \mathsf{Type\text{-}2} \wedge a_e = a_d \wedge |I(C^{(d)}, C^{(e)})| = 0\right]$ is upper-bounded. Let $i = \max I(A^{(d)}, A^{(e)})$. Then, under the condition **B8**,

  $$H^{(e)} = H^{(d)} \Rightarrow V_i^{(e)} = V_i^{(d)}.$$

  If $i = 1$, then

  $$V_1^{(e)} = \delta^{(a)}(0^b, A_1^{(e)}) \text{ and } V_1^{(d)} = \delta^{(a)}(0^b, A_1^{(d)}).$$

On the contrary, $A_1^{(e)} \neq A_1^{(d)}$ is satisfied[6] and the condition **B8** with the conditions in (11) (thus the last condition in **B8**, $A \neq A' \wedge |A| = b \wedge W = W' \wedge |A'| < b$, is ignored) imply

$$V_1^{(e)} = \delta^{(a)}(0^b, A_1^{(e)}) \neq \delta^{(a)}(0^b, A_1^{(d)}) = V_1^{(d)}.$$

If $i \geq 2$, then

$$V_i^{(e)} = V_i^{(d)} \Leftrightarrow \delta^{(a)}\left(W_{i-1}^{(e)}, A_i^{(e)}\right) = \delta^{(a)}\left(W_{i-1}^{(d)}, A_i^{(d)}\right).$$

By $A_i^{(d)} \neq A_i^{(e)}$ and the condition **B8** with the conditions in (11), to satisfy the above equation, $W_{i-1}^{(d)} \neq W_{i-1}^{(e)}$ should be satisfied. As $W_{i-1}^{(d)}$ is chosen uniformly at random from $\{0,1\}^b \backslash \left\{W_{i-1}^{(e)}\right\}$ and $\delta^{(d)}\left(\cdot, A_i^{(d)}\right)$ is bijective from the condition **B7**, we have $p_{1,1} \leq 1/(2^b - 1)$.

- $p_{1,2} = \Pr\left[H^{(d)} = H^{(e)} \middle| \mathsf{Type\text{-}2} \wedge a_e \neq a_d \wedge |I(C^{(d)}, C^{(e)})| = 0\right]$ is upper-bounded. Thus, the following equation is considered.

$$\delta^{(a)}\left(W_{a_d-1}^{(d)}, A_{a_d}^{(d)}\right) = H^{(d)} = H^{(e)} = \delta^{(a)}\left(W_{a_e-1}^{(e)}, A_{a_e}^{(e)}\right).$$

By $a_e \neq a_d$, the tweaks corresponding to the TRP outputs $W_{a_d-1}^{(d)}$ and $W_{a_e-1}^{(e)}$ are distinct. Thus, $W_{a_d-1}^{(d)}$ and $W_{a_e-1}^{(e)}$ are independently chosen, and at least one of them is chosen uniformly at random from $\{0,1\}^b$. (Note that for $x \in \{d, e\}$, if $a_x = 1$, then $H^{(x)} = \delta^{(a)}\left(0^b, A_1^{(x)}\right)$.) Under the condition **B7**, at least one of $\delta^{(a)}\left(W_{a_d-1}^{(d)}, A_{a_d}^{(d)}\right)$ and $\delta^{(a)}\left(W_{a_e-1}^{(e)}, A_{a_e}^{(e)}\right)$ are uniformly distributed over $\{0,1\}^b$. Hence, we have $p_{1,2} \leq 1/2^b$.

- $p_2 = \Pr\left[S^{(d)} = S^{(e)} \middle| \mathsf{Type\text{-}2} \wedge |I(C^{(d)}, C^{(e)})| \geq 1\right]$ is upper bounded. Let $i = \max I(C^{(d)}, C^{(e)})$. Note that under the $\mathsf{Type\text{-}2}$ decryption query, $\ell_e = \ell_d$ is satisfied. Subsequently, under the condition **B6**,

$$S_1^{(d)} = S_1^{(e)} \Leftrightarrow X_{i+1}^{(d)} = X_{i+1}^{(e)} \Leftrightarrow \delta^{(d)}\left(Y_i^{(d)}, C_i^{(d)}\right) = \delta^{(e)}\left(Y_i^{(e)}, M_i^{(e)}\right),$$

where $M_i^{(e)} = \gamma_{|C_i^{(e)}|}^{(d)}(\mathsf{msb}_{|C_i^{(e)}|}(Y_i^{(e)}), C_i^{(e)})$. $C_i^{(d)} \neq C_i^{(e)}$ and the condition **B6** with (12) imply $Y_i^{(d)} \neq Y_i^{(e)}$, and thus, we have $X_i^{(d)} \neq X_i^{(e)}$. Hence,

$$p_2 \leq \Pr\Big[\delta^{(e)}\left(Y_i^{(e)}, M_i^{(e)}\right) = \delta^{(d)}\left(Y_i^{(d)}, C_i^{(d)}\right) \\ \Big| \mathsf{Type\text{-}2} \wedge X_i^{(d)} \neq X_i^{(e)} \wedge |I(C^{(d)}, C^{(e)})| \geq 1\Big].$$

By $X_i^{(d)} \neq X_i^{(e)}$, $Y_i^{(d)}$ is chosen uniformly at random from $\{0,1\}^b \backslash \{Y_i^{(e)}\}$. As $\delta^{(d)}\left(\cdot, C_i^{(d)}\right)$ is bijective from the condition **B5**, we have $p_2 \leq 1/(2^b - 1)$.

The above-mentioned upper bounds yield the following:

$$\Pr\left[S^{(d)} = S^{(e)} \middle| \mathsf{Type\text{-}2}\right] \leq \frac{1}{2^b - 1}. \tag{13}$$

[6] Note that $A_1^{(e)} = \varepsilon \wedge A_1^{(d)} \neq \varepsilon$, $A_1^{(e)} \neq \varepsilon \wedge A_1^{(d)} = \varepsilon$, or $A_1^{(e)} \neq \varepsilon \wedge A_1^{(d)} \neq \varepsilon$.

5 Implementation

The performance of PFB is evaluated through concrete hardware implementations. For the lightweight TBC, we use a variant of SKINNY having 64-bit block length and 192-bit tweakey, i.e., SKINNY-64-192 [BJK+16]. Our focus is on the hardware implementations as it is one of the most important targets for lightweight cryptography, and a significant improvement is expected in threshold implementation, as will be discussed in Section 5.3. Although evaluation in other platforms is beyond the scope of this paper, the proposed method is expected to yield a good performance on microcontrollers similar to that of SAEB designed with the same design criteria [NMSS18].

Comparison The hardware performance is compared with that of a state-of-the-art alternative having the same level of security: SAEB [NMSS18] instantiated with a lightweight block cipher GIFT-128-128 [BPP+17]. Another possible option is to instantiate SAEB with SKINNY-128-128 thereby aligning the primitives between the two modes of operation. However, this comparison can favor the proposed method over SAEB, because a tweakable block cipher is currently less efficient compared to a block cipher, as we will see in this section, and thus, we selected GIFT over SKINNY.

Notation In the following, SKINNY-64-192 and GIFT-128-128 are simply referred to as SKINNY and GIFT. In addition, a mode of operation M instantiated with a primitive P is described as M[P].

Design Policy For a fair comparison, PFB[SKINNY] and SAEB[GIFT] are implemented under the same design policy. They are designed as co-processors aiming at accelerating the main time-consuming part of AD processing, encryption, and decryption. Meanwhile, the co-processors expect an external controller for handling special cases such as padding and final-block processing. To avoid a hidden cost, the designs hold a key, nonce, and tweak during their lifetimes. In other words, there is no need for storing them in external registers and feeding them multiple times. This policy affects the implementation of on-the-fly key scheduling, as we will see in the next section. The circuit area has the highest priority in optimization. The designs are described by a hardware description language (HDL) at the register-transfer level (RTL). We do not make netlist-level optimization, except for the scan flip-flops commonly used for compact implementations [MPL+11]; the standard cells for scan flip-flops are explicitly instantiated in HDL. For SCA-protected implementations, we consider TI secured up to the first-order attacks. The implementations have simple register interfaces: they do not have a standard bus interface, such as the AXI bus.

5.1 PFB[SKINNY]

SKINNY uses three distinct 64-bit states, namely **TK1**, **TK2**, and **TK3**, for a tweakey schedule. In this particular design, **TK3** stores a 64-bit tweak. The remaining **TK1** and **TK2** store a 128-bit secret key.

Figure 4 depicts the hardware architecture of PFB[SKINNY]. As shown in Figure 4, PFB[SKINNY] is realized as a thin wrapper of the SKINNY implementation; the additional components are 4-bit XOR, selector, and AND gate only.

The SKINNY implementation follows the conventional nibble-serial architecture [BJK+16], but the tweakey-schedule implementation is designed from scratch. The implementations called the **TK1**, **TK2**, and **TK3** arrays are based on a common architecture comprising an array of scan flip-flops and integrated on-the-fly key scheduling [MPL+11], as shown in Figure 4. However, the changes made by the on-the-fly key scheduling should be reverted

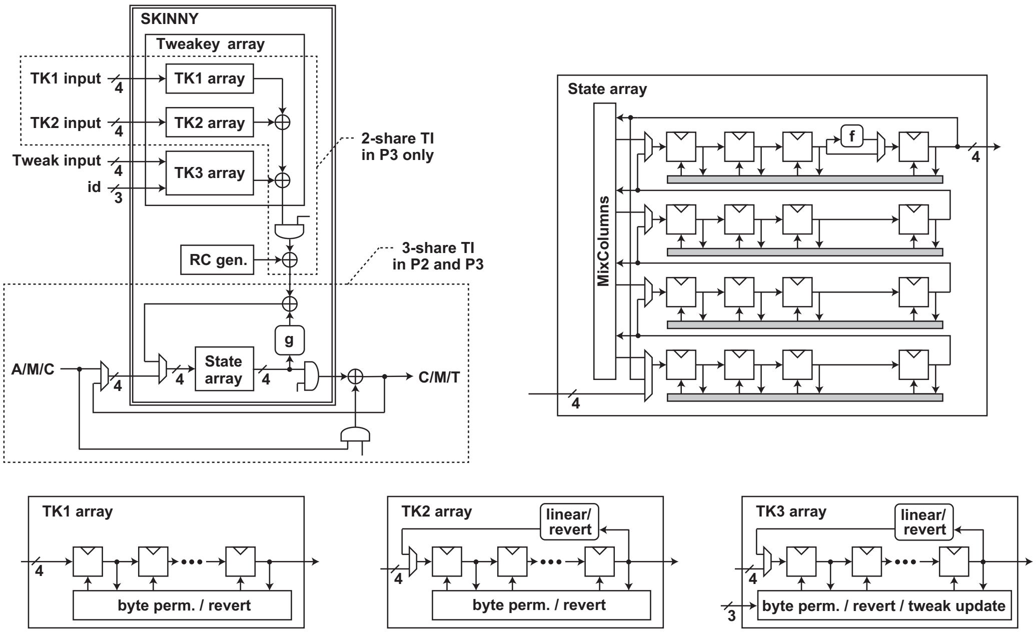

Figure 4: PFB[SKINNY] hardware architecture.

to begin the next TBC call without feeding the same key again. As SKINNY schedules **TK1**, **TK2**, and **TK3** by a nibble permutation and a nibble-wise linear transformation for each round, we can obtain efficient inverse maps that revert the final tweakey state to the initial one. Such inverse maps are integrated to the **TK1**, **TK2**, and **TK3** arrays along with forward on-the-fly scheduling.

Based on Eq. (3), the 64-bit tweak is given by $\mathsf{id}\|N\|\mathsf{ctr}$: a 3-bit number distinguishing the operations $\mathsf{id} = \mathsf{str}_3(i)$, 45-bit nonce N, and a current block number realized by a 16-bit counter $\mathsf{ctr} = \mathsf{str}_{16}(j)$. id and ctr are updated for each TBC call. For an efficient computation, the **TK3** array integrates the circuit for (i) changing id and (ii) incrementing and clearing the counter ctr. Using the above functionality, a user needs to feed $\mathsf{id}\|N\|\mathsf{ctr}$ only once for a given nonce N.

A single SKINNY round uses 16 cycles, and thus, SKINNY comprising 40 rounds finishes in $16 \times 40 = 640$ cycles. We need an additional cycle for updating a tweak stored in the **TK3** array for the next TBC call. Consequently, a 64-bit message or ciphertext block is consumed in 641 cycles.

5.2 SAEB[GIFT]

Figure 5 depicts the hardware architecture of SAEB[GIFT]. The overall architecture is based on the conventional design [NMSS18], but the shift registers for synchronization are removed by considering the design policy. It is also realized as a thin wrapper of the underlying GIFT implementation.

The GIFT implementation is based on the nibble-serial architecture [BPP$^+$17], but the key array is redesigned to efficiently revert the changes made by on-the-fly key scheduling. Similar to SKINNY, GIFT has a linear key scheduling algorithm, and thus, we can obtain an efficient inverse map that reverts the final key state to the initial one. The key array is designed with a 32-bit datapath to efficiently integrate the inverse key-schedule map (the function block labeled with "revert"), as shown in Figure 5.

The S-box is split into two stages, namely g and f, for TI following the conventional

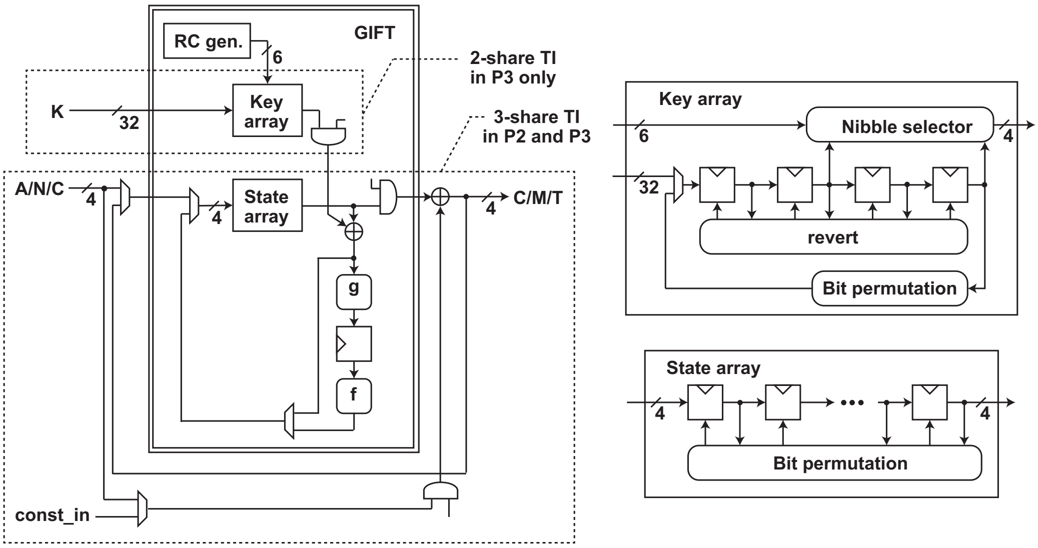

Figure 5: SAEB[GIFT] hardware architecture.

work [GJC+17]. Consequently, a single GIFT round uses 33 cycles for 32 S-box lookups and one pipeline latency. Consequently, the 40-round operation of GIFT requires $33 \times 40 = 1,320$ cycles.

5.3 Threshold Implementation

There is an option between protected and unprotected key/tweakey schedules. Conventional attacks, such as differential power analysis (DPA) [KJJ99], cannot be used to attack the key schedule that is independent of an attacker-controllable input, e.g., plaintext or ciphertext. That is generally not true for TBCs, but SKINNY has the same property as far as the attacker-controllable tweak is placed in **TK3**, which is scheduled independently of **TK1** and **TK2**. Consequently, some previous studies prioritized circuit area and used unprotected key-schedule implementations [BJK+16, PMK+11, UHA17]. Meanwhile, if we consider a profiling attack on the key/tweakey schedule, it is also reasonable to choose a protected key-schedule implementation. Considering the cost-security trade-off, we implement PFB[SKINNY] and SAEB[GIFT] with three profiles: (**P1**) the unprotected implementation, (**P2**) TI with the unprotected key schedule, and (**P3**) TI with the protected key schedule.

Table 2 summarizes the number of registers needed for the SKINNY and GIFT implementations for the different profiles. In (**P1**), both SKINNY and GIFT use 256 bits in total. In (**P2**), on the other hand, SKINNY uses fewer registers, i.e., 384 bits, compared to 512 bits, because of the smaller block length. SKINNY still performs better in (**P3**) because of efficient sharing of the key/tweakey schedule. As both GIFT and SKINNY have linear key/tweakey schedules, they can be realized with only two shares. Moreover, there is no need for protecting **TK3** of SKINNY, which stores a public tweak. Consequently, SKINNY and GIFT use 512 and 684 bits in (**P3**), respectively.

We use the formulae for the 3-share uniform S-boxes for SKINNY and GIFT from the conventional work [BJK+16] and [GJC+17], respectively. TI is implemented by duplicating the state/key/tweakey arrays and replacing the decomposed S-boxes (f and g) with their shared maps. Figure 4 and 5 show the boundaries of sharing for each profile.

Table 2: Number of registers for implementing SKINNY and GIFT in different profiles.

Target	Profile	TI/State	TI/Key	State	Tweak/key	Total
SKINNY	(**P1**)	—	—	64	192	256
GIFT	(**P1**)	—	—	128	128	256
SKINNY	(**P2**)	✓	—	192	192	384
GIFT	(**P2**)	✓	—	384	128	512
SKINNY	(**P3**)	✓	✓	192	320	512
GIFT	(**P3**)	✓	✓	384	256	640

Table 3: Breakdown of the post-synthesis circuit area of PFB[SKINNY] and SAEB[GIFT].

Target	Component	Circuit area [GE]		
		(**P1**)	(**P2**)	(**P3**)
PFB[SKINNY]	Total	3,111	4,492	5,858
	Total/SKINNY	2,956	4,284	5,649
	Total/SKINNY/State array	532	1,757	1,757
	Total/SKINNY/Tweakey array	2,062	2,062	3,419
SAEB[GIFT]	Total	2,761	5,037	6,229
	Total/GIFT	2,541	4,756	5,947
	Total/GIFT/State array	975	2,925	2,925
	Total/GIFT/Key array	1201	1,226	2,410

5.4 Performance Evaluation and Comparison

The designs are synthesized with the NanGate 45-nm standard cell library [Nan] using Synopsys Design Compiler while preserving the module hierarchy. Table 3 details the breakdown of the post-synthesis performances.

We first discuss the unprotected implementations (**P1**). The circuit areas of PFB[SKINNY] and SAEB[GIFT] are $3,111$ and $2,761$ [GE], respectively. SKINNY and GIFT dominate the circuit areas of PFB[SKINNY] and SAEB[GIFT]. The additional costs for the mode of operations are limited. The sizes of the state and key arrays are almost proportional to their register sizes, e.g., the 64-bit SKINNY state array (532 [GE]) is almost half the size of the 128-bit GIFT state array (975 [GE]).

Although the PFB[SKINNY] implementation is larger than that of SAEB[GIFT] by 350 [GE], this is a positive result because (i) GIFT is known to have a superior performance than SKINNY [BJK+16] and (ii) lightweight TBC is an emerging technology compared to a lightweight block cipher. Note also that PFB[SKINNY] is twice as fast as SAEB[GIFT]: PFB[SKINNY] and SAEB[GIFT] consume a 64-bit message/ciphertext block using 640 and $1,320$ cycles, respectively. Moreover, PFB has parallelizable encryption, as discussed in Section 3.

Table 4 presents a performance comparison with previous implementations. The unprotected implementations of SAEB[GIFT] and PFB[SKINNY] are smaller than the previous implementations of AES-based AEs (SAEB[AES128] [NMSS18], CLOC[AES128], SILC[AES128], OTR[AES128] [BBM16]). The bit-serial Ascon implementation without an interface has a smaller circuit area of $2,570$ [GE] [GWDE15]; however, the implementation needs an additional 128-bit key register to run another encryption/decryption with the same key. If we add the size of the key register (640 [GE] for 5 [GE/bit]) to $2,570$ [GE], the Ascon implementation has a similar circuit size compared to that of PFB[SKINNY]. In addition, the Ascon implementation with an interface including a 128-bit key register has

Table 4: Performance comparison; latency is that of a single call of a primitive (block cipher, tweakable block cipher, or permutation).

Target	TI	Area [GE]	Latency [cycles]	Security [bits]	Standard-cell library	Ref.
PFB[SKINNY] (**P1**)	—	3,111	641	64	NanGate 45-nm	**Ours**
SAEB[GIFT] (**P1**)	—	2,761	1,320	64	NanGate 45-nm	**Ours**
SAEB[AES128]	—	3,502	231	64	NanGate 45-nm	[NMSS18]
CLOC[AES128]	—	4,310	210	64	STMicro. 90-nm	[BBM16]
SILC[AES128]	—	4,220	210	64	STMicro. 90-nm	[BBM16]
OTR[AES128]	—	6,770	210	64	STMicro. 90-nm	[BBM16]
Ascon w/o IF	—	2,570	3,072	128	UMC 90-nm	[GWDE15]
Ascon w/ IF	—	3,750	3,072	128	UMC 90-nm	[GWDE15]
Deoxys (Round*)	—	11,936	14	128	UMC 180-nm	[JNPS16]
Ketje-JR	—	5,447	16	96	NanGate 45-nm	[ANR18]
PFB[SKINNY] (**P2**)	✓	4,492	641	64	NanGate 45-nm	**Ours**
PFB[SKINNY] (**P3**)	✓	5,858	641	64	NanGate 45-nm	**Ours**
SAEB[GIFT] (**P2**)	✓	5,037	1,320	64	NanGate 45-nm	**Ours**
SAEB[GIFT] (**P3**)	✓	6,229	1,320	64	NanGate 45-nm	**Ours**
Ascon w/o IF	✓	7,970	3,072	128	UMC 90-nm	[GWDE15]
Ascon w/ IF	✓	9,190	3,072	128	UMC 90-nm	[GWDE15]
Ketje-JR	✓	18,335	16	96	NanGate 45-nm	[ANR18]

3,750 [GE].

We then discuss the protected implementations. With (**P2**), the PFB[SKINNY] implementation uses 4,492 [GE], which is smaller than that of SAEB[GIFT] (5,037 [GE]). This is explained by the fewer registers summarized in Table 2. PFB[SKINNY] is still advantageous with (**P3**): the circuit areas of PFB[SKINNY] and SAEB[GIFT] are 5,858 and 6,229 [GE], respectively. The protected PFB implementations are smaller than those of Ascon [GWDE15]) and Ketje [ANR18] in conventional work, as presented in Table 4. This can also be explained by the number of registers. The sponge-based AEs have a relatively large state (384 bits for Ascon and 200 bits for Ketje-JR) that should be protected with three shares.

In summary, the unprotected PFB[SKINNY] implementation is competitive against the unprotected SAEB[GIFT] implementations and other conventional implementations. The benefit of a small block length, enabled by PFB, becomes even larger with TI, where the number of registers are multiplied as shown in Table 2. Consequently, the protected PFB[SKINNY] implementation outperforms those of SAEB[GIFT], Ascon [GWDE15], and Ketje [ANR18].

Acknowledgments

The authors would like to thank the anonymous reviewers for their helpful comments and suggestions.

References

[AMP12] Elena Andreeva, Bart Mennink, and Bart Preneel. The parazoa family: generalizing the sponge hash functions. *Int. J. Inf. Sec.*, 11(3):149–165, 2012.

[ANR18] Victor Arribas, Svetla Nikova, and Vincent Rijmen. Guards in Action: First-Order SCA Secure Implementations of Ketje Without Additional Randomness. In *DSD 2018*, pages 492–499. IEEE Computer Society, 2018.

[BBI+15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A Block Cipher for Low Energy. In *ASIACRYPT 2015*, volume 9453 of *LNCS*, pages 411–436. Springer, 2015.

[BBM16] Subhadeep Banik, Andrey Bogdanov, and Kazuhiko Minematsu. Low-area hardware implementations of CLOC, SILC and AES-OTR. In *HOST 2016*, pages 71–74. IEEE Computer Society, 2016.

[BCG+12] 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*, volume 7658 of *LNCS*, pages 208–225. Springer, 2012.

[BDPA07] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge Functions. In *Ecrypt Hash Workshop 2007*, 2007.

[BDPA11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications. In *SAC 2011*, volume 7118 of *LNCS*, pages 320–337. Springer, 2011.

[BJK+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 *CRYPTO 2016*, volume 9815 of *LNCS*, pages 123–153. Springer, 2016.

[BKL+07] 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*, volume 4727 of *LNCS*, pages 450–466. Springer, 2007.

[BL16] Karthikeyan Bhargavan and Gaëtan Leurent. On the Practical (In-)Security of 64-bit Block Ciphers: Collision Attacks on HTTP over TLS and OpenVPN. In *CCS 2016*, pages 456–467. ACM, 2016.

[BPP+17] 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 *CHES 2017*, volume 10529 of *LNCS*, pages 321–345. Springer, 2017.

[BSS+13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK Families of Lightweight Block Ciphers. *IACR Cryptology ePrint Archive*, 2013:404, 2013.

[CIMN17] Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, and Mridul Nandi. Blockcipher-Based Authenticated Encryption: How Small Can We Go? In *CHES 2017*, volume 10529 of *LNCS*, pages 277–298. Springer, 2017.

[GJC+17] Naina Gupta, Arpan Jati, Anupam Chattopadhyay, Somitra Kumar Sanadhya, and Donghoon Chang. Threshold Implementations of GIFT: A Trade-off Analysis. *IACR Cryptology ePrint Archive*, 2017:1040, 2017.

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED Block Cipher. In *CHES 2011*, volume 6917 of *LNCS*, pages 326–341. Springer, 2011.

[GWDE15] Hannes Groß, Erich Wenger, Christoph Dobraunig, and Christoph Ehrenhöfer. Suit up! - Made-to-Measure Hardware Implementations of ASCON. In *DSD 2015*, pages 645–652. IEEE Computer Society, 2015.

[IKMP19a] Tetsu Iwata, Mustafa Khairallah, Kazuhiko Minematsu, and Thomas Peyrin. Duel of the Titans: The Romulus and Remus Families of Lightweight AEAD Algorithms. *IACR Cryptology ePrint Archive*, 2019:992, 2019.

[IKMP19b] Tetsu Iwata, Mustafa Khairallah, Kazuhiko Minematsu, and Thomas Peyrin. Romulus v1.0, round 1 candidate of the lightweight crypto standardization process, 2019.

[IMPS17] Tetsu Iwata, Kazuhiko Minematsu, Thomas Peyrin, and Yannick Seurin. ZMAC: A Fast Tweakable Block Cipher Mode for Highly Secure Message Authentication. In *CRYPTO 2017*, volume 10403 of *LNCS*, pages 34–65. Springer, 2017.

[Iwa06] Tetsu Iwata. New Blockcipher Modes of Operation with Beyond the Birthday Bound Security. In *FSE 2006*, volume 4047 of *LNCS*, pages 310–327. Springer, 2006.

[Iwa08] Tetsu Iwata. Authenticated Encryption Mode for Beyond the Birthday Bound Security. In *AFRICACRYPT 2008*, volume 5023 of *LNCS*, pages 125–142. Springer, 2008.

[JNPS16] Jérémy Jean, Ivica Nikolic, Thomas Peyrin, and Yannick Seurin. Deoxys v1.41. Submitted to the CAESAR competition. 2016.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In *CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer, 1999.

[KR11] Ted Krovetz and Phillip Rogaway. The Software Performance of Authenticated-Encryption Modes. In *FSE 2011*, volume 6733 of *LNCS*, pages 306–327. Springer, 2011.

[LN17] Eik List and Mridul Nandi. Revisiting Full-PRF-Secure PMAC and Using It for Beyond-Birthday Authenticated Encryption. In *CT-RSA 2017*, volume 10159 of *LNCS*, pages 258–274. Springer, 2017.

[LRW02] Moses Liskov, Ronald L. Rivest, and David A. Wagner. Tweakable Block Ciphers. In *CRYPTO 2002*, volume 2442 of *LNCS*, pages 31–46. Springer, 2002.

[LS13] Rodolphe Lampe and Yannick Seurin. Tweakable Blockciphers with Asymptotically Optimal Security. In *FSE 2013*, volume 8424 of *LNCS*, pages 133–151. Springer, 2013.

[LST12] Will Landecker, Thomas Shrimpton, and R. Seth Terashima. Tweakable Blockciphers with Beyond Birthday-Bound Security. In *CRYPTO 2012*, volume 7417 of *LNCS*, pages 14–30. Springer, 2012.

[MI15] Kazuhiko Minematsu and Tetsu Iwata. Tweak-Length Extension for Tweakable Blockciphers. In *IMACC 2015*, volume 9496 of *LNCS*, pages 77–93. Springer, 2015.

[MI17] Kazuhiko Minematsu and Tetsu Iwata. Cryptanalysis of PMACx, PMAC2x, and SIVx. *IACR Trans. Symmetric Cryptol.*, 2017(2):162–176, 2017.

[Min14] Kazuhiko Minematsu. Parallelizable Rate-1 Authenticated Encryption from Pseudorandom Functions. In *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 275–292. Springer, 2014.

[MPL$^+$11] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 69–88. Springer, 2011.

[MV04] David A. McGrew and John Viega. The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In *INDOCRYPT 2004*, volume 3348 of *LNCS*, pages 343–355. Springer, 2004.

[Nan] NanGate. NanGate FreePDK45 open cell library. http://www.nangate.com.

[NIS18] NIST. Submission requirements and evaluation criteria for the lightweight cryptography standardization process. Available at https://csrc.nist.gov/Projects/lightweight-cryptography, 2018.

[NMSS18] Yusuke Naito, Mitsuru Matsui, Takeshi Sugawara, and Daisuke Suzuki. SAEB: A Lightweight Blockcipher-Based AEAD Mode of Operation. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):192–217, 2018.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold Implementations Against Side-Channel Attacks and Glitches. In *ICICS 2006*, volume 4307 of *LNCS*, pages 529–545. Springer, 2006.

[NRS11] Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches. *J. Cryptology*, 24(2):292–321, 2011.

[NS19] Yusuke Naito and Takeshi Sugawara. Lightweight authenticated encryption mode of operation for tweakable block ciphers. *IACR Cryptology ePrint Archive*, 2019:339, 2019.

[PGV93] Bart Preneel, René Govaerts, and Joos Vandewalle. Hash Functions Based on Block Ciphers: A Synthetic Approach. In *CRYPTO'93*, volume 773 of *LNCS*, pages 368–378. Springer, 1993.

[PMK$^+$11] Axel Poschmann, Amir Moradi, Khoongming Khoo, Chu-Wee Lim, Huaxiong Wang, and San Ling. Side-Channel Resistant Crypto for Less than 2, 300 GE. *J. Cryptology*, 24(2):322–345, 2011.

[PS16] Thomas Peyrin and Yannick Seurin. Counter-in-Tweak: Authenticated Encryption Modes for Tweakable Block Ciphers. In *CRYPTO 2016*, volume 9814 of *LNCS*, pages 33–63. Springer, 2016.

[SIH$^+$11] Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. Piccolo: An Ultra-Lightweight Blockcipher. In *CHES 2011*, volume 6917 of *LNCS*, pages 342–357. Springer, 2011.

[SMMK13] Tomoyasu Suzaki, Kazuhiko Minematsu, Sumio Morioka, and Eita Kobayashi. TWINE: A Lightweight Block Cipher for Multiple Platforms. In *SAC 2012*, volume 7707 of *LNCS*, pages 339–354. Springer, 2013.

[SSA+07] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-Bit Blockcipher CLEFIA (Extended Abstract). In *FSE 2007*, volume 4593 of *LNCS*, pages 181–195. Springer, 2007.

[Sta09] Martijn Stam. Blockcipher-Based Hashing Revisited. In *FSE 2009*, volume 5665 of *LNCS*, pages 67–83. Springer, 2009.

[UHA17] Rei Ueno, Naofumi Homma, and Takafumi Aoki. Toward More Efficient DPA-Resistant AES Hardware Architecture Based on Threshold Implementation. In *COSADE 2017*, volume 10348 of *LNCS*, pages 50–64. Springer, 2017.

Appendix

A PFB Functions $\gamma_l^{(e)}, \gamma_l^{(d)}, \delta^{(e)}, \delta^{(d)}, \delta^{(a)}$ Satisfy B2-B8

We show that the functions of PFB given in Section 3.2 satisfy the conditions **B2-B8**. The functions $\gamma_l^{(e)}, \gamma_l^{(d)}, \delta^{(e)}, \delta^{(d)}$ are of the following forms:

$$\begin{aligned}
\gamma_l^{(e)}(Y, M) &= Y \oplus M, \text{ where } 0 < l \leq b, \text{ and } Y, M \in \{0,1\}^l \\
\gamma_l^{(d)}(Y, C) &= Y \oplus C, \text{ where } 0 < l \leq b, \text{ and } Y, C \in \{0,1\}^l \\
\delta^{(e)}(Y, M) &= \mathsf{ozp}_b(M) \oplus \left(0^{|M|} \| \mathsf{lsb}_{b-|M|}(Y)\right), \text{ where } Y \in \{0,1\}^b, M \in \{0,1\}^{\leq b}. \\
\delta^{(d)}(Y, C) &= Y \oplus \mathsf{ozp}_b(C), \text{ where } Y \in \{0,1\}^b, C \in \{0,1\}^{\leq b}. \\
\delta^{(a)}(W, A) &= W \oplus \mathsf{ozp}_b(A), \text{ where } Y \in \{0,1\}^b, A \in \{\varepsilon\} \cup \{0,1\}^{\leq b}.
\end{aligned}$$

Condition B2

Clearly, for any $Y \in \{0,1\}^l$, $\gamma_l^{(e)}(Y, \cdot)$ is bijective and $\gamma_l^{(d)}(Y, \cdot)$ is the inverse of $\gamma_l^{(e)}(Y, \cdot)$.

Condition B3

Let $M \in \{0,1\}^l$ and $Y \in \{0,1\}^b$. Subsequently,

$$\begin{aligned}
\delta^{(d)}(Y, \gamma_l^{(e)}(\mathsf{msb}_l(Y), M)) &= Y \oplus \mathsf{ozp}_b\left(\gamma_l^{(e)}(\mathsf{msb}_l(Y), M)\right) \\
&= Y \oplus \mathsf{ozp}_b\left(\mathsf{msb}_l(Y) \oplus M\right) \\
&= \mathsf{ozp}_b(M) \oplus \left(0^l \| \mathsf{lsb}_{b-l}(Y)\right).
\end{aligned}$$

Hence, we have $\delta^{(e)}(Y, M) = \delta^{(d)}(Y, \gamma_l^{(e)}(\mathsf{msb}_l(Y), M))$.

Condition B4

Clearly, for any $M \in \{0,1\}^l$, $\gamma_l^{(e)}(\cdot, M)$ is bijective.

Condition B5

Distinctly, for any $C \in \{0,1\}^{\leq b}$, $\delta^{(d)}(\cdot, C)$ is bijective.

Condition B6

The contraposition of the condition **B6** is considered, i.e.,

$$\begin{aligned}&(C = C' \wedge Y \neq Y')\\ \vee&(C \neq C' \wedge Y = Y' \wedge |C| < b \wedge |C'| < b)\\ \vee&(C \neq C' \wedge Y = Y' \wedge |C| = b \wedge |C'| = b) \Rightarrow \delta^{(e)}(Y, \gamma^{(d)}_{|C|}(\mathsf{msb}_{|C|}(Y), C)) \neq \delta^{(d)}(Y', C').\end{aligned}$$

For $Y \in \{0,1\}^b, C \in \{0,1\}^{\leq b}$, the functions are of the following forms:

$$\begin{aligned}\delta^{(e)}(Y, \gamma^{(d)}_{|C|}(\mathsf{msb}_{|C|}(Y), C)) &= \mathsf{ozp}_b(\gamma^{(d)}_{|C|}(\mathsf{msb}_{|C|}(Y), C)) \oplus \left(0^{|C|} \| \mathsf{lsb}_{b-|C|}(Y)\right)\\ &= \mathsf{ozp}_b(\mathsf{msb}_{|C|}(Y) \oplus C)) \oplus \left(0^{|C|} \| \mathsf{lsb}_{b-|C|}(Y)\right)\\ &= Y \oplus \mathsf{ozp}_b(C),\\ \delta^{(d)}(Y', C') &= Y' \oplus \mathsf{ozp}_b(C').\end{aligned}$$

Hence, the condition **B6** is satisfied.

Condition B7

Evidently, for any $A \in \{0,1\}^{\leq b}$, $\delta^{(a)}(\cdot, A)$ is bijective.

Condition B8

The contraposition of the condition **B8** is considered, i.e.,

$$\begin{aligned}&(A = A' \wedge W \neq W')\\ \vee&(A \neq A' \wedge W = W' \wedge |A| < b \wedge |A'| < b)\\ \vee&(A \neq A' \wedge W = W' \wedge |A| = b \wedge |A'| = b) \Rightarrow \delta^{(a)}(W, A) \neq \delta^{(a)}(W, A').\end{aligned}$$

Clearly, the function $\delta^{(a)}(W, A)$ satisfies the condition **B8**.

B PFB

PFB is given in Algorithm 2 and shown in Figure 6.

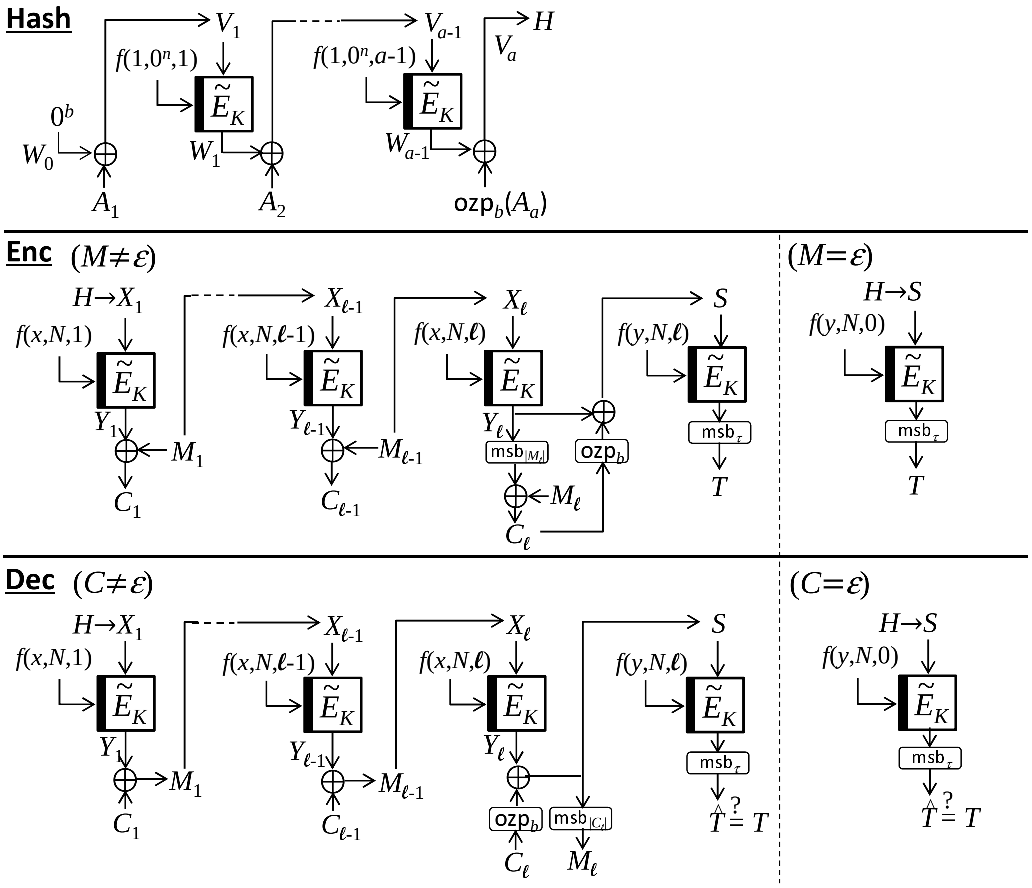

Figure 6: PFB. $A_1,\dots,A_a \xleftarrow{b} A$. $M_1,\dots,M_\ell \xleftarrow{b} M$ (in the encryption algorithm) and $C_1,\dots,C_\ell \xleftarrow{b} C$ (in the decryption algorithm). If $A \neq \varepsilon \wedge |A| \mod b = 0$, then $x \leftarrow 2$; else, $x \leftarrow 3$. If $C \neq \varepsilon \wedge |C| \mod b = 0$, then $y \leftarrow x + 2$; else, $y \leftarrow x + 4$.

Algorithm 2 PFB

Encryption $\mathsf{PFB.Enc}[\widetilde{E}_K](N, A, M)$

```
 1: $X_1 \leftarrow \mathsf{PFB.Hash}[\widetilde{E}_K](A)$
 2: **if** $A \neq \varepsilon \wedge |A| \mod b = 0$ **then** $x \leftarrow 2$; **else** $x \leftarrow 3$
 3: **if** $M = \varepsilon$ **then** $C \leftarrow \varepsilon$; $\ell \leftarrow 0$; goto step 9
 4: $M_1, \ldots, M_\ell \xleftarrow{b} M$
 5: **for** $i = 1, \ldots, \ell$ **do**
 6:     $Y_i \leftarrow \widetilde{E}_K^{f(x,N,i)}(X_i)$; $C_i \leftarrow \mathsf{msb}_{|M_i|}(Y_i) \oplus M_i$
 7:     $X_{i+1} \leftarrow \mathsf{ozp}_b(M_i) \oplus \left(0^{|M_i|} \| \mathsf{lsb}_{b-|M_i|}(Y_i)\right)$
 8: **end for**
 9: **if** $M \neq \varepsilon \wedge |M| \mod b = 0$ **then** $y \leftarrow x + 2$; **else** $y \leftarrow x + 4$
10: $S \leftarrow X_{\ell+1}$; $T \leftarrow \mathsf{msb}_\tau \left(\widetilde{E}_K^{f(y,N,\ell)}(S)\right)$; $C \leftarrow C_1 \| \cdots \| C_\ell$
11: **return** $(C, T)$
```

Decryption $\mathsf{PFB.Dec}[\widetilde{E}_K](N, A, C, T)$

```
 1: $X_1 \leftarrow \mathsf{PFB.Hash}[\widetilde{E}_K](A)$
 2: **if** $A \neq \varepsilon \wedge |A| \mod b = 0$ **then** $x \leftarrow 2$; **else** $x \leftarrow 3$
 3: **if** $C = \varepsilon$ **then** $M \leftarrow \varepsilon$; $\ell \leftarrow 0$; goto step 8
 4: $C_1, \ldots, C_\ell \xleftarrow{b} C$
 5: **for** $i = 1, \ldots, \ell$ **do**
 6:     $Y_i \leftarrow \widetilde{E}_K^{f(x,N,i)}(X_i)$; $M_i \leftarrow \mathsf{msb}_{|C_i|}(Y_i) \oplus C_i$; $X_{i+1} \leftarrow Y_i \oplus \mathsf{ozp}_b(C_i)$
 7: **end for**
 8: **if** $C \neq \varepsilon \wedge |C| \mod b = 0$ **then** $y \leftarrow x + 2$; **else** $y \leftarrow x + 4$
 9: $S \leftarrow X_{\ell+1}$; $\hat{T} \leftarrow \mathsf{msb}_\tau \left(\widetilde{E}_K^{f(y,N,\ell)}(S)\right)$; $M \leftarrow M_1 \| \cdots \| M_\ell$
10: **if** $T = \hat{T}$ **then return** $M$; **else return** $\perp$
```

Hash $\mathsf{PFB.Hash}[\widetilde{E}_K](A)$

```
 1: $A_1, \ldots, A_a \xleftarrow{b} A$; $W_0 \leftarrow 0^b$
 2: **for** $i = 1, \ldots, a-1$ **do** $V_i \leftarrow W_{i-1} \oplus \mathsf{ozp}_b(A_i)$; $W_i \leftarrow \widetilde{E}_K^{f(1,0^n,i)}(V_i)$
 3: $V_a \leftarrow W_{a-1} \oplus \mathsf{ozp}_b(A_a)$; $H \leftarrow V_a$
 4: **return** $H$
```