GIFT-COFB

Subhadeep Banik[1], Avik Chakraborti[2], Tetsu Iwata[3], Kazuhiko Minematsu[4], Mridul Nandi[5], Thomas Peyrin[6,7], Yu Sasaki[2], Siang Meng Sim[6] and Yosuke Todo[2]

[1] LASEC, Ecole Polytechnique Fédérale de Lausanne, Switzerland
[2] NTT Secure Platform Laboratories, Japan
[3] Nagoya University, Japan
[4] NEC Corporation, Japan
[5] Indian Statistical Institute, Kolkata, India
[6] Nanyang Technological University, Singapore
[7] Temasek Laboratories@NTU, Singapore

giftcofb@googlegroups.com
https://www.isical.ac.in/~lightweight/COFB/

Abstract. In this article, we propose GIFT-COFB, an Authenticated Encryption with Associated Data (AEAD) scheme, based on the GIFT lightweight block cipher and the COFB lightweight AEAD operating mode. We explain how these two primitives can fit together and the various design adjustments possible for performance and security improvements. We show that our design provides excellent performances in all constrained scenarios, hardware or software, while being based on a provably-secure mode and a well analysed block cipher.

Keywords: GIFT · COFB · authenticated encryption · lightweight · lower bound

1 Introduction

Confidentiality and authentication are two critical security properties, historically offered with separated cryptographic components. However, due to the possible security issues that might arise when combining these two components and in a hope for performance gains, so-called authenticated encryption (AE) is now becoming more prominent. AE is a symmetric-key cryptographic scheme providing both confidentiality and authenticity in a single primitive. In 2002, Rogaway [20] proposed the concept of Authenticated Encryption with Associated Data (AEAD), well adopted nowadays, which allows in addition a user to authenticate some associated data, without encrypting it (typically some Internet packet header).

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. After a decade of many advances in the field of lightweight symmetric-key cryptography, an extremely lightweight block cipher – GIFT [3] and a very low state size AEAD scheme – COFB [8] were concurrently proposed at CHES 2017 conference. The former is an ad-hoc primitive while the latter is an operating mode, but both primarily focus on obtaining very good hardware implementation results. GIFT reduces the footprint of its algorithmic operations to the bare minimum without compromising its security (actually improving it when compared to PRESENT cipher [7], probably the most famous lightweight block cipher). On the other hand, COFB minimises the additional state required for a rate-1 block cipher based AEAD scheme. It was then very natural to match these two primitives to build a very efficient candidate for the NIST lightweight cryptography competition. Yet, several details need to be handled when matching, in order to maintain the full performance and ensure compliance with NIST requirements.

In this work, we describe the GIFT-COFB authenticated encryption, which instantiates the COFB (COmbined FeedBack) block cipher based AEAD mode with the GIFT block cipher, but with several small tweaks on both COFB and GIFT to further improve their efficiency. Here, we consider the overhead in size, thus the state memory size beyond the underlying block cipher itself (including the key schedule) as one of the main criteria we want to minimize, which is particularly relevant for hardware implementations.

This version supports all the desirable properties mentioned in the NIST lightweight cryptography portfolio [14], and it is efficient for lightweight implementations as well.

There are many approaches for designing a secure and lightweight block cipher based AEAD. We focus on using the lightweight, very efficient and well analyzed block cipher GIFT-128 [3] and minimizing the total encryption/decryption state size by using combined feedback over the block cipher output and the data blocks along with a tweak dependent secret masking (as used in XEX [21]). This combination helps us to minimize the amount of masking by a factor of 2 from [21].

The COFB mode achieves several interesting features. It provides a high rate of 1 (i.e, it needs only one block cipher call per input block). The mode is inverse-free, as it does not need a block cipher inverse during decryption or decryption. In addition to these features, this mode has a very small state size, namely $1.5n + k$ bits, where n and k denote the underlying block cipher block size and key size respectively.

Our Contributions. In this article, we describe GIFT-COFB, an Authenticated Encryption with Associated Data (AEAD) scheme, based on the GIFT-128 lightweight block cipher and the COFB lightweight AEAD operating mode. We analyse how these two primitives can be adapted to fit together and how various design adjustments that we made to improve performance and security. We recall that COFB is a provably secure operating mode and that GIFT block cipher has been thoroughly analysed by its designers and retains a very comfortable security margin even after a lot of third party analysis. We show that our design provides excellent performances in all constrained scenarios, both hardware and software.

Organisation of the paper. We first introduce some notations in Section 2 and describe our proposal GIFT-COFB in Section 3. Then, we explain the design rationale in Section 4 and recall security analysis conducted on the mode COFB and on the internal primitive GIFT in Section 5. Finally, we report latest hardware and software implementation results in Sections 6 and 7.

2 Preliminaries

2.1 Notation

For any $X \in \{0,1\}^*$, where $\{0,1\}^*$ is the set of all finite bit strings (including the empty string ϵ), we denote the number of bits of X by $|X|$. Note that $|\epsilon| = 0$. For a string X and an integer $t \leq |X|$, $\mathsf{Trunc}_t(X)$ is the first t bits of X. Throughout this document, n represents the block size in bits of the underlying block cipher E_K. Typically, we consider $n = 128$ and GIFT-128 is the underlying block cipher, where K is the 128-bit GIFT-128 key. 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). Note that ϵ is considered as an incomplete block and $\epsilon \in \mathcal{B}^<$. 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:

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

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

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

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 $Z = \epsilon$, we let $z = 1$ and $Z[1] = \epsilon$. We write $||Z|| = z$ (number of blocks present in Z). 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.

2.2 Underlying Finite Field $\mathbb{F}_{2^n}$

Let $\mathbb{F}_{2^s}$ denote the binary Galois field of size 2^s, for a positive integer s. Field addition and multiplication between $a, b \in \mathbb{F}_{2^s}$ are represented by $a \oplus b$ (or $a + b$ whenever understood) and $a \cdot b$ respectively. Any field element $a \in \mathbb{F}_{2^s}$ can be represented by any of the following equivalent ways for $a_0, a_1, \ldots, a_{s-1} \in \{0,1\}$.

- An s-bit string $a_{s-1} \cdots a_0 \in \{0,1\}^s$.
- A polynomial $a(x) = a_0 + a_1 x + \cdots + a_{s-1} x^{s-1}$ of degree at most $(s-1)$.

2.3 Choice of Primitive Polynomials

In our construction, the primitive polynomial [1] used to represent the field $\mathbb{F}_{2^{64}}$ is

$$p_{64}(x) = x^{64} + x^4 + x^3 + x + 1.$$

We denote the primitive element $0^{s-2}10 \in \mathbb{F}_{2^s}$ by α_s (here $s = 64$). We use α to mean α_s for notational simplicity. The field multiplication $a(x) \cdot b(x)$ is the polynomial $r(x)$ of degree at most $(s-1)$ such that $a(x)b(x) \equiv r(x) \mod p_s(x)$.

Multiplication by Primitive Element α. We first see an example how we can multiply by $\alpha = \alpha_{64}$. Multiplying an element $b := b_{63} b_{62} \cdots b_0 \in \mathbb{F}_{2^{64}}$ by the primitive element α of $\mathbb{F}_{2^{64}}$ can be done very efficiently as follows:

$$b \cdot \alpha = \begin{cases} b \ll 1, & \text{if } b_{63} = 0, \\ (b \ll 1) \oplus 0^{59}11011, & \text{else,} \end{cases}$$

where $(b \ll r)$ denotes left shift of b by r bits. For $b \in \mathbb{F}_{2^{64}}$, we use $2 \cdot b$ (or $2^m \cdot b$) and $3 \cdot b$ (or $3^m \cdot b$) to denote $\alpha \cdot b$ (or $\alpha^m \cdot b$) and $(1+\alpha) \cdot b$ (or $(1+\alpha)^m \cdot b$), respectively.

2.4 Authenticated Encryption and Security Definitions

An authenticated encryption or AE algorithm takes a nonce N (which is a value never repeats at encryption) together with associated date A and plaintext M, the encryption function of AE, $\mathcal{E}_K$, produces a tagged-ciphertext (C, T) where $|C| = |M|$ and $|T| = t$. It provides both privacy of a plaintext $M \in \{0,1\}^*$ and authenticity or integrity of M as well as associate data $A \in \{0,1\}^*$. The corresponding decryption function, $\mathcal{D}_K$, takes (N, A, C, T) and returns a decrypted plaintext M when the verification on (N, A, C, T) is successful, otherwise returns the atomic error symbol denoted by $\perp$.

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

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

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

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

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

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

to denote the maximum forging advantage for all adversaries running in time t, making q 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 [10, 22]. Let $\mathcal{A}$ be an adversary that only makes non-repeating queries to $\mathcal{D}_K$. Then, we define the AE-advantage of $\mathcal{A}$ against $\mathcal{E}$ as

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

where $\perp$-oracle always returns $\perp$ and \$-oracle is as the privacy advantage. We similarly define $\mathbf{Adv}^{\mathrm{AE}}_{\mathcal{E}}((q, q_f), (\sigma, \sigma_f), t) = \max_{\mathcal{A}} \mathbf{Adv}^{\mathrm{AE}}_{\mathcal{E}}(\mathcal{A})$, 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.

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

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

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

Coefficients-H Technique. Coefficients-H technique was developed by Patarin, that is a convenient tool for bounding the advantage (see [15, 9]). We will use this technique (without giving a proof) to prove our security claims. Consider two oracles $\mathcal{O}_0 = (\$, \perp)$ (the ideal oracle) and $\mathcal{O}_1$ (the real oracle, i.e., our construction). Let $\mathcal{V}$ denotes 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 oracle (or ideal oracle, respectively). 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 queries and 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, e.g., in [9]. We will later use this result to prove the security of our construction in Theorem 1 by defining certain $\mathcal{V}_{\text{good}}$ for our games, and evaluating the bounds, ϵ_1 and ϵ_2.

3 Specification

3.1 Syntax

The encryption algorithm (with authentication), denoted as $\mathsf{GIFT\text{-}COFB}(K, N, A, M) \mapsto (C, T)$, takes as input an encryption key $K \in \{0,1\}^{128}$, a nonce $N \in \{0,1\}^{128}$, associated data $A \in \{0,1\}^*$, and a message $M \in \{0,1\}^*$. The nonce N can include a counter to make the nonce non-repeating. It generates a ciphertext $C \in \{0,1\}^{|M|}$ and a tag $T \in \{0,1\}^{128}$.

The decryption algorithm (with verification), denoted as $\mathsf{GIFT\text{-}COFB}^{-1}(K, N, A, C, T) \mapsto M$, takes (K, N, A, C, T) as input. It generates a message $M \in \{0,1\}^{|C|}$ or a special symbol $\perp$ denoting rejection.

3.2 Building Blocks of GIFT-COFB

3.2.1 Building Blocks of COFB

Block Cipher. The underlying encryption cipher, E_K, is an 128-bit block cipher with 128-bit key equivalent to GIFT-128 but with a small tweak in the input and output data format. See Section 3.2.2 for the specification and Section 4.2 for the rationale.

Padding Function. For $x \in \{0,1\}^*$, we define padding function Pad as

$$\mathsf{Pad}(x) = \begin{cases} x & \text{if } x \neq \epsilon \text{ and } |x| \bmod n = 0 \\ x \| 10^{(n - (|x| \bmod n) - 1)} & \text{otherwise.} \end{cases}$$

Note that $\mathsf{Pad}(\epsilon) = 10^{n-1}$.

Feedback Function. Let $Y \in \{0,1\}^{128}$ and $(Y[1], Y[2]) \xleftarrow{64} Y$, where $Y[i] \in \{0,1\}^{64}$. We define $G : \{0,1\}^{128} \to \{0,1\}^{128}$ as

$$G(Y) = (Y[2], Y[1] \lll 1),$$

where for a string X, $X \lll r$ is the left rotation of X by r bits. We also view G as the 128×128 non-singular binary matrix, so we write $G(Y)$ and $G \cdot Y$ interchangeably. For $M \in \{0,1\}^{128}$ and $Y \in \{0,1\}^{128}$, we define $\rho_1(Y, M) = G \cdot Y \oplus M$. The feedback function ρ and its corresponding ρ' are defined as

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

Note that when $(X, M) = \rho'(Y, C)$ then $X = (G \oplus I) \cdot Y \oplus C$, where I is the 128×128 identity matrix. Our choice of G ensures that $G \oplus I$ has rank $n - 1$ (precisely, 127, in our construction with $n = 128$). When Y is chosen randomly, both $\rho_1(Y, M)$ (during encryption) and $\rho_1(Y, Y \oplus C)$ (during decryption) also has almost full entropy.

We need this property when we bound probability of bad events later.

Tweak Value for The Last Block. Given the last block of associated data, $A \in \{0,1\}^*$, we define $\delta_A \in \{1, 2\}$ as follows:

$$\delta_A = \begin{cases} 1 & \text{if } A \neq \epsilon \text{ and } n \text{ divides } |A| \\ 2 & \text{otherwise.} \end{cases}$$

Given the last block of either a message or a ciphertext, $Z \in \{0,1\}^*$, we define $\delta_Z \in \{1, 2\}$ as follows:

$$\delta_Z = \begin{cases} 1 & \text{if } n \text{ divides } |Z| \\ 2 & \text{otherwise.} \end{cases}$$

This will be used to differentiate the cases that the last block of A or Z is n bits or shorter, for Z being a message or a ciphertext. We also define a formatting function Fmt for a pair of bit strings (A, Z). 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

$$\begin{cases} \big((A[1], \mathsf{t}[1]), \ldots, (\overline{A[a]}, \mathsf{t}[a])\big) & \textit{if } Z = \epsilon \\ \big((A[1], \mathsf{t}[1]), \ldots, (\overline{A[a]}, \mathsf{t}[a]), (Z[1], \mathsf{t}[a+1]), \ldots, (\overline{Z[z]}, \mathsf{t}[a+z])\big) & \textit{if } Z \neq \epsilon \end{cases}$$

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.

Lemma 1. *The function* $\mathsf{Fmt}(\cdot)$ *is prefix-free.*

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

3.2.2 GIFT building blocks

Initialization and Finalization. The 128-bit plaintext P is loaded into the cipher state S which will be expressed as 4 32-bit segments, $S = \{S_0, S_1, S_2, S_3\}$, where $S_i \in \{0,1\}^{32}$. On the other hand, the 128-bit secret key K is loaded into the key state KS which will be expressed as 8 16-bit words, $KS = \{W_0, W_1, \ldots, W_7\}$, where $W_i \in \{0,1\}^{16}$.

$$\mathsf{Initalize}(P) = \begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{bmatrix} \leftarrow \begin{bmatrix} B_0 & \| & B_1 & \| & B_2 & \| & B_3 \\ B_4 & \| & B_5 & \| & B_6 & \| & B_7 \\ B_8 & \| & B_9 & \| & B_{10} & \| & B_{11} \\ B_{12} & \| & B_{13} & \| & B_{14} & \| & B_{15} \end{bmatrix},$$

$$\mathsf{Initalize}(K) = \begin{bmatrix} W_0 & \| & W_1 \\ W_2 & \| & W_3 \\ W_4 & \| & W_5 \\ W_6 & \| & W_7 \end{bmatrix} \leftarrow \begin{bmatrix} B_0\|B_1 & \| & B_2\|B_3 \\ B_4\|B_5 & \| & B_6\|B_7 \\ B_8\|B_9 & \| & B_{10}\|B_{11} \\ B_{12}\|B_{13} & \| & B_{14}\|B_{15} \end{bmatrix},$$

where B_i are the arriving bytes.

The function Finalize will be the reverse process, outputting the state byte by byte.

SubCells Function. We denote the SubCells function $S \leftarrow \mathsf{SubCells}(S)$ as the following set of instructions:

$$\begin{aligned}
S_1 &\leftarrow S_1 \oplus (S_0 \ \&\ S_2) \\
S_0 &\leftarrow S_0 \oplus (S_1 \ \&\ S_3) \\
S_2 &\leftarrow S_2 \oplus (S_0 \mid S_1) \\
S_3 &\leftarrow S_3 \oplus S_2 \\
S_1 &\leftarrow S_1 \oplus S_3 \\
S_3 &\leftarrow \sim S_3 \\
S_2 &\leftarrow S_2 \oplus (S_0 \ \&\ S_1) \\
\{S_0, S_1, S_2, S_3\} &\leftarrow \{S_3, S_1, S_2, S_0\},
\end{aligned}$$

where $\&, |$ and $\sim$ are `AND`, `OR` and `NOT` operation respectively.

PermBits Function. We define the parsing of S_i into 32 individual bits as

$$(S_i[31], S_i[30], \ldots, S_i[0]) \xleftarrow{1} S_i.$$

We denote

$$\mathsf{PermBits}(S) = \{Pb_0(S_0), Pb_1(S_1), Pb_2(S_2), Pb_3(S_3)\},$$

where Pb_i is described in Table 1, the row "Index" shows the indexing of the 32 bits in all S_i's and the row "S_i" shows the ending position of the bits. For example, $S_1[1]$ (the 2nd rightmost bit) is shifted 1 position to the right, to the initial position of $S_1[0]$, while $S_1[0]$ is shifted 8 positions to the left where $S_1[8]$ was.

AddRoundKey Function. We define the AddRoundKey function AddRoundKey as

$$\mathsf{AddRoundKey}(S, KS, i) = \{S_0, S_1 \oplus (W_6 \parallel W_7), S_2 \oplus (W_2 \parallel W_3), S_3 \oplus \mathtt{Const_i}\},$$

where $\mathtt{Const_i} = \mathtt{0x800000XY}$ is the i-th round constant and the byte $\mathtt{XY} = 00c_5c_4c_3c_2c_1c_0$ is the round constant generated using the a 6-bit affine LFSR, whose state is updated as follows:

$$c_5\|c_4\|c_3\|c_2\|c_1\|c_0 \leftarrow c_4\|c_3\|c_2\|c_1\|c_0\|c_5 \oplus c_4 \oplus 1.$$

Table 1: Specifications of bit permutation Pb_i.

Index	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Pb_0	29	25	21	17	13	9	5	1	30	26	22	18	14	10	6	2
Pb_1	30	26	22	18	14	10	6	2	31	27	23	19	15	11	7	3
Pb_2	31	27	23	19	15	11	7	3	28	24	20	16	12	8	4	0
Pb_3	28	24	20	16	12	8	4	0	29	25	21	17	13	9	5	1

Index	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Pb_0	31	27	23	19	15	11	7	3	28	24	20	16	12	8	4	0
Pb_1	28	24	20	16	12	8	4	0	29	25	21	17	13	9	5	1
Pb_2	29	25	21	17	13	9	5	1	30	26	22	18	14	10	6	2
Pb_3	30	26	22	18	14	10	6	2	31	27	23	19	15	11	7	3

The six bits, c_i, are initialized to zero, and updated *before* being used in a given round.

The values of the constants for each round are given in the table below, encoded to byte values for each round, with c_0 being the least significant bit.

Rounds	**Constants**
1 - 16	`01,03,07,0F,1F,3E,3D,3B,37,2F,1E,3C,39,33,27,0E`
17 - 32	`1D,3A,35,2B,16,2C,18,30,21,02,05,0B,17,2E,1C,38`
33 - 48	`31,23,06,0D,1B,36,2D,1A,34,29,12,24,08,11,22,04`

Key State Update Function. The key state update function KeyUpdate is defined as follows:

$$\mathsf{KeyUpdate}(KS) = \{W_6 \ggg 2, W_7 \ggg 12, W_0, W_1, W_2, W_3, W_4, W_5\}$$

3.3 GIFT-COFB Pseudocode

We present the specifications of GIFT-COFB in Fig. 1, where α and $(1+\alpha)$ are written as 2 and 3. See also Fig. 2. The encryption and decryption algorithms are denoted by COFB-$\mathcal{E}_K$ and COFB-$\mathcal{D}_K$. We remark that the nonce length is 128 bits, which is enough for the security up to the birthday bound. The nonce is processed as $E_K(N)$ to yield the first internal chaining value. The encryption algorithm takes A and M, and outputs C and T such that $|C| = |M|$ and $|T| = 128$. The decryption algorithm takes (N, A, C, T) and outputs M or $\perp$. Both encryption and decryption algorithms use block cipher E_K and the key K is implicitly given to them.

4 Design Rationale

As both GIFT and COFB are already well-established primitives, in this section we explain the rationale for this combination, followed by the tweaks we made to these original publications to enhance the performance and security.

4.1 AEAD Scheme: GIFT-COFB

COFB is a block cipher based authenticated encryption mode that uses GIFT-128 as the underlying block cipher and GIFT-COFB can be viewed as an efficient integration of the

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

$Y[0] \leftarrow E_K(N),\ L \leftarrow \textsf{Trunc}_{n/2}(Y[0])$
$(A[1], \ldots, A[a]) \xleftarrow{n} \textsf{Pad}(A)$
if $M \neq \epsilon$ **then**
  $(M[1], \ldots, M[m]) \xleftarrow{n} \textsf{Pad}(M)$
for $i = 1$ **to** $a - 1$
  $L \leftarrow 2 \cdot L$
  $X[i] \leftarrow A[i] \oplus G \cdot Y[i-1] \oplus L \| 0^{n/2}$
  $Y[i] \leftarrow E_K(X[i])$
if $|A| \bmod n = 0$ **and** $A \neq \epsilon$ **then** $L \leftarrow 3 \cdot L$
else $L \leftarrow 3^2 \cdot L$
if $M = \epsilon$ **then** $L \leftarrow 3^2 \cdot L$
$X[a] \leftarrow A[a] \oplus G \cdot Y[a-1] \oplus L \| 0^{n/2}$
$Y[a] \leftarrow E_K(X[a])$
for $i = 1$ **to** $m - 1$
  $L \leftarrow 2 \cdot L$
  $C[i] \leftarrow M[i] \oplus Y[i+a-1]$
  $X[i+a] \leftarrow M[i] \oplus G \cdot Y[i+a-1] \oplus L \| 0^{n/2}$
  $Y[i+a] \leftarrow E_K(X[i+a])$
if $M \neq \epsilon$ **then**
  if $|M| \bmod n = 0$ **then** $L \leftarrow 3 \cdot L$
  else $L \leftarrow 3^2 \cdot L$
  $C[m] \leftarrow M[m] \oplus Y[a+m-1]$
  $X[a+m] \leftarrow M[m] \oplus G \cdot Y[a+m-1] \oplus L \| 0^{n/2}$
  $Y[a+m] \leftarrow E_K(X[a+m])$
  $C \leftarrow \textsf{Trunc}_{|M|}(C[1] \| \ldots \| C[m])$
  $T \leftarrow \textsf{Trunc}_{\tau}(Y[a+m])$
else $C \leftarrow \epsilon$, $T \leftarrow \textsf{Trunc}_{\tau}(Y[a])$
return (C, T)

Algorithm $E_K(X)$

$S \leftarrow \textsf{Initialize}(X)$
$KS \leftarrow \textsf{Initialize}(K)$
for $i = 1$ **to** 40
  $S \leftarrow \textsf{SubCells}(S)$
  $S \leftarrow \textsf{PermBits}(S)$
  $S \leftarrow \textsf{AddRoundKey}(S, KS, i)$
  $KS \leftarrow \textsf{KeyUpdate}(KS)$
$Y \leftarrow \textsf{Finalize}(S)$
return Y

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

$Y[0] \leftarrow E_K(N),\ L \leftarrow \textsf{Trunc}_{n/2}(Y[0])$
$(A[1], \ldots, A[a]) \xleftarrow{n} \textsf{Pad}(A)$
if $C \neq \epsilon$ **then**
  $(C[1], \ldots, C[c]) \xleftarrow{n} \textsf{Pad}(C)$
for $i = 1$ **to** $a - 1$
  $L \leftarrow 2 \cdot L$
  $X[i] \leftarrow A[i] \oplus G \cdot Y[i-1] \oplus L \| 0^{n/2}$
  $Y[i] \leftarrow E_K(X[i])$
if $|A| \bmod n = 0$ **and** $A \neq \epsilon$ **then** $L \leftarrow 3 \cdot L$
else $L \leftarrow 3^2 \cdot L$
if $C = \epsilon$ **then** $L \leftarrow 3^2 \cdot L$
$X[a] \leftarrow A[a] \oplus G \cdot Y[a-1] \oplus L \| 0^{n/2}$
$Y[a] \leftarrow E_K(X[a])$
for $i = 1$ **to** $c - 1$
  $L \leftarrow 2 \cdot L$
  $M[i] \leftarrow Y[i+a-1] \oplus C[i]$
  $X[i+a] \leftarrow M[i] \oplus G \cdot Y[i+a-1] \oplus L \| 0^{n/2}$
  $Y[i+a] \leftarrow E_K(X[i+a])$
if $C \neq \epsilon$ **then**
  if $|C| \bmod n = 0$ **then**
    $L \leftarrow 3 \cdot L$
    $M[c] \leftarrow Y[a+c-1] \oplus C[c]$
  else
    $L \leftarrow 3^2 \cdot L$, $c' \leftarrow |C| \bmod n$
    $M[c] \leftarrow \textsf{Trunc}_{c'}(Y[a+c-1] \oplus C[c]) \| 10^{n-c'-1}$
  $X[a+c] \leftarrow M[c] \oplus G \cdot Y[a+c-1] \oplus L \| 0^{n/2}$
  $Y[a+c] \leftarrow E_K(X[a+c])$
  $M \leftarrow \textsf{Trunc}_{|C|}(M[1] \| \ldots \| M[c])$
  $T' \leftarrow \textsf{Trunc}_{\tau}(Y[a+c])$
else $M \leftarrow \epsilon$, $T' \leftarrow \textsf{Trunc}_{\tau}(Y[a])$
if $T' = T$ **then return** M, **else return** $\perp$

Figure 1: The encryption and decryption algorithms of GIFT-COFB.

COFB and GIFT-128. GIFT-128 maintains an 128-bit state and 128-bit key. To be precise, GIFT is a family of block ciphers parametrized by the state size and the key size and all the members of this family are lightweight and can be efficiently deployed on lightweight applications. COFB mode on the other hand, computes of "COmbined FeedBack" (of block cipher output and data block) to uplift the security level. This actually helps us to design a mode with low state size and eventually to have a low state implementation. This technique actually resist the attacker to control the input block and next block cipher

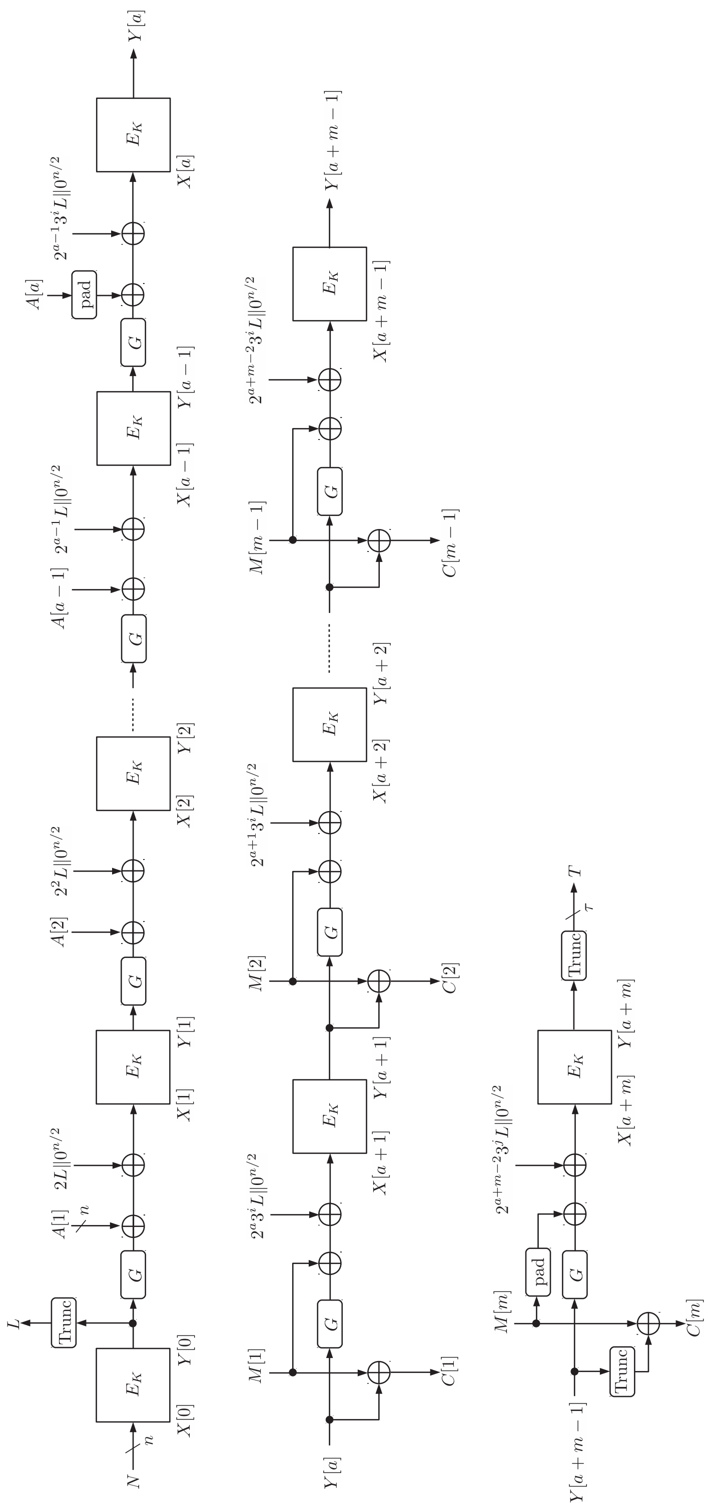

Figure 2: Encryption of COFB. In the rightmost figure, the case of encryption for empty M (hence a MAC for (N, A)) can be highlighted as $T = \mathsf{Trunc}_\tau(Y[a])$

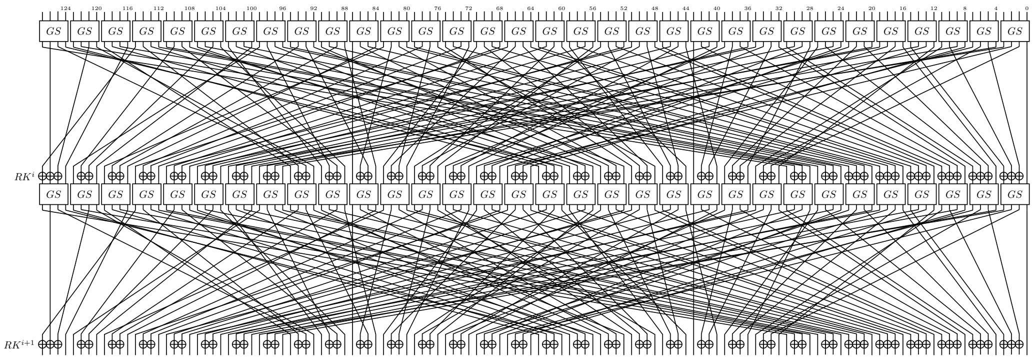

Figure 3: 2 rounds of GIFT-128.

input simultaneously. Overall, a combination of GIFT and COFB can be considered to be one of the most efficient lightweight, low state block cipher based AEAD construction.

4.2 Underlying Block Cipher: GIFT

GIFT-128 is an 128-bit Substitution-Permutation network (SPN) based block cipher with a key length of 128-bit. It is a 40-round iterative block cipher with identical round function. For brevity, we simply call it GIFT.

There are different ways to perceive GIFT-128, the more pictorial description is detailed in Section 2 of [4], which looks like a larger version of PRESENT cipher with 32 4-bit S-boxes and an 128-bit bit permutation (see Figure 3). In our work, we use an alternative description of GIFT, using bitslice description which is similar to Appendix A of [4]. Note that the security properties are equivalent up to bit arrangement of the plaintext and ciphertext.

GIFT is considered to be one of the lightest design existing in the literature. It is denoted as "Small PRESENT" as the design rationale of GIFT follows that of PRESENT [7]. However, GIFT has got rid of several well known weaknesses existing in PRESENT with regards to linear cryptanalysis. Overall GIFT promises much increased efficiency (both lighter and faster) over PRESENT. GIFT is a very simple design that outperforms even SIMON and SKINNY for round based implementations. It consists of very simple operations such that the total hardware footprint is almost consumed by the underlying and the cipher storage. The design is somewhat "optimal" as a weaker S-box (than GIFT S-box) would lead to a weaker design. The linear layer is completely free for a round-based implementation in hardware (consisting of simply bit-wiring) and the constants are generated thanks to a very lightweight LFSR. The key schedule is also very light, simply consisting of shifts. The presented security analysis details and hardware implementation results also support the claims made by the designers.

Although there is almost no impact on hardware implementation, there are several motivations for using bitslice implementation (non-LUT based) instead of LUT based implementation of GIFT when we consider software implementation. Here, we will state the 3 most obvious benefits relating to its 3 steps in a round function.

Constant time non-linear layer. For LUT based implementation, we can consider updating 2 GIFT S-boxes (1 byte) in a single memory call with a reasonable 256 entries LUT. This would require 16 lookups and it takes approximately 16 to 64 cycles to do all S-boxes in a round, assuming a few cycles to access the RAM. Using bitslice implementation, it requires just 11 basic operations (or 10 with XNOR operation) to compute all the S-boxes in parallel. And more importantly, using bitslice implementation has the nice feature that it doesn't need any RAM and that it is constant time, mitigating potential timing attacks.

Efficient linear layer. While it is basically free on hardware, for software implementation it is extremely slow and complex to implement. This effect can be reduced by doing several blocks in parallel using none other than bitslice implementation. Even for a single block encryption, bitslice implementation is still more efficient that LUT based implementation because of the way the bits are packed.

Simpler key addition. For LUT based implementation, the subkeys need to be XORed to bit positions that are 3 bits apart, making the key addition tedious and non-trivial. An option is to precompute the subkeys, but even so the key addition would require several XOR operations to update the 128-bit state. Using bitslice, the bits that were once 3 bits apart are now packed together in 32-bit words, making the key addition as simple as just 2 XOR operations.

4.3 Authenticated Encryption Mode: COFB

COFB is a lightweight AEAD mode. The mode presented in this write up differs slightly with the original proposal. They are as follows.

- We change the nonce to be 128 bits.
- We change the feedback (more precisely the G matrix) to make it more hardware efficient.
- We now deal with empty data. We change the mask update function for the purpose.
- We change the padding for the associated data. To be precise, if the associated data is empty, then padding the associated data will yield the constant block 10^{n-1} (n: block cipher state size).

We observed that, the updates make the design more lightweight and more efficient to deal with short data inputs. However, these updates do not have impact on the security of the mode (only a nominal 1-bit security degradation).

5 Security

5.1 Security proof of COFB

We present the security analysis of COFB in Theorem 1.

Theorem 1 (Main Theorem)**.**

$$\mathbf{Adv}^{\mathrm{AE}}_{\mathsf{COFB}}((q,q_f),(\sigma,\sigma_f),t) \leq \mathbf{Adv}^{\mathrm{prp}}_{\mathsf{GIFT}}(q',t') + \frac{\binom{q'}{2}}{2^n} + \frac{1}{2^{n/2}} + \frac{q_f(n+4)}{2^{n/2+1}}$$

$$+ \frac{3\sigma^2 + q_f + (q+\sigma+\sigma_f)\cdot\sigma_f}{2^n}$$

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

Proof. We assume $q' \leq 2^{\frac{n}{2}-1}$. We make a transition by using an n-bit (uniform) random permutation P instead of E_K, which is GIFT, and next an n-bit (uniform) random function R instead of P. The first two terms in our bounds comes from these two transitions using the standard PRP-PRF switching lemma and the computation to the information security reduction (e.g., see [5]).

Thus we only need a bound for COFB with R, denoted by COFB-R. Here, we prove

$$\mathbf{Adv}^{\mathsf{AE}}_{\mathsf{COFB\text{-}R}}((q,q_f),(\sigma,\sigma_f),\infty) \le \frac{1}{2^{n/2}} + \frac{q_f(n+4)}{2^{n/2+1}} + \frac{2\sigma^2 + q_f + (q+\sigma+\sigma_f)\cdot\sigma_f}{2^n}. \quad (1)$$

Let (N_i, A_i, M_i) and (C_i, T_i) denote the i-th encryption query and response respectively for $1 \le i \le q$. We use the notation $(A_i[1], \ldots, A_i[a_i]) \xleftarrow{n} \mathsf{Pad}(A_i)$, $(M_i[1], \ldots, M_i[m_i]) \xleftarrow{n} \mathsf{Pad}(M_i)$ and $(C_i[1], \ldots, C_i[m_i]) \xleftarrow{n} \mathsf{Pad}(C_i)$. Let $\ell_i = a_i + m_i + 1$, which denotes the total input block length (including nonce) for the i-th encryption query. The i-th decryption query is $(N_i^*, A_i^*, C_i^*, T_i^*)$ with a response Z_i^* (either $\perp$ for an invalid decryption attempt or a message). We similarly define c_i^* and a_i^*, and write $\ell_i^* = a_i^* + c_i^* + 1$. We have $\sigma = \sum_i \ell_i$ and $\sigma_f = \sum_i \ell_i^*$. We also use the notation $(L_i[j], R_i[j]) \xleftarrow{n/2} X_i[j]$ for all $i \in [1..q]$ and $j \in [1..\ell_i]$.

Real Oracle. Real oracle follows COFB-R (where E_K is replaced by R). We use $X_i[j]$ (resp. $Y_i[j]$) for $i = 1, \ldots, q$ and $j = 0, \ldots, \ell_i$ for the j-th input (resp. output) of the internal R invoked during the i-th encryption query, with the order of invocation shown in Fig. 1. We set $X_i[0] = N_i$ and $Y_i[\ell_i] = T_i$. We write $L_i = \mathsf{Trunc}_{n/2}(Y_i[0])$.

The following relaxations are introduced that only gain the advantage. After making all the encryption queries and forging attempts, release all the Y-values for the encryption queries only. The transcript due to encryption queries consists of $(N_i, A_i, M_i, Y_i)_i$ where Y_i denotes $(Y_i[0], \ldots, Y_i[\ell_i]) = Y_i[0..\ell_i]$.

Ideal Oracle. 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 $C_i[j] = Y_i[j-1] \oplus M_i[j]$, 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 $Z_i^* = \perp$ (here we assume that the adversary makes only fresh queries).

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

$$\tau = ((N_i, A_i, M_i, Y_i)_{i \in \{1,\ldots,q\}}, (N_{i'}^*, A_{i'}^*, C_{i'}^*, T_{i'}^*, Z_{i'}^*)_{i' \in \{1,\ldots,q_f\}}).$$

Note that, X_i-values of encryption queries are also uniquely determined following construction based on N_i, A_i, M_i and Y_i.

Definition of p_i and i'. For the i-th decryption query, we define $p_i = -1$ if there is no j with $N_j = N_i^*$. In this case i' is not defined. Otherwise, there is a unique index i' with $N_{i'} = N_i^*$. We define p_i as the length of the longest common prefix of $\mathsf{Fmt}(A_i^*, C_i^*)$ and $\mathsf{Fmt}(A_{i'}, C_{i'})$. Since Fmt is prefix-free, it holds that $p_i < \min\{\ell_i^*, \ell_{i'}\}$.

Bad Views. 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: $X_{i_1}[j_1] = X_{i_2}[j_2]$ for some $(i_1, j_1) \ne (i_2, j_2)$ where $j_1, j_2 > 0$.

B2: $Y_{i_1}[j_1] = Y_{i_2}[j_2]$ for some $(i_1, j_1) \ne (i_2, j_2)$ where $j_1, j_2 > 0$.

B3: $\mathsf{mcoll}(R) > n/2$ where R is the tuple of all $R_i[j]$ values.

B4: $X_i^*[p_i + 1] = X_{i_1}[j_1]$ for some (i, i_1, j_1) with $j_1 \ne 0$.

B5: $p_i = \ell_i^* - 1$ and $X_i^*[p_i + 1] = X_{i_1}[j_1]$ for some (i, i_1, j_1) with $Y_{i_1}[j_1] = T_i^*$.

B6: $p_i \neq -1$ and $X_i^*[p_i+1] = X_{i'}[0]$ for some (i, i').

B7: $p_i \neq -1, \ell_i^* - 1$ and $X_i^*[p_i+1] = X_{i_1}[0]$ and $X_i^*[p_i+2] = X_{i_2}[j_2]$ for some $i_1 \neq i'$ and (i_2, j_2).

B8: For some i, $Z_i^* \neq \perp$. This clearly cannot happen for the ideal oracle case.

We add some intuitions on these events. When **B1** does not hold, then 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.

B2 event is an auxiliary event which is required to bound **B5**.

Similarly, **B3** would be required to bound the probability of the other bad events. 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. So, when **B3** 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. The event **B4** to **B7** are different cases for which (p_i+1)-st input for the i-th forging attempt are not fresh.

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 the ideal oracle (this will complete the first condition of the Coefficients-H technique).

Lemma 2.

$$\Pr_{\mathsf{ideal}}[\tau \notin \mathcal{V}_{\text{good}}] \leq \frac{1}{2^{n/2}} + \frac{q_f(n+4)}{2^{n/2+1}} + \frac{2\sigma^2}{2^n}$$

Proof of Lemma 2. 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.

(1) $\Pr[\mathbf{B1}] \leq \sigma^2/2^{n+1}$: For any $(i_1, j_1) \neq (i_2, j_2)$ with $j_1, j_2 \geq 1$, the equality event $X_{i_1}[j_1] = X_{i_2}[j_2]$ has a probability at most 2^{-n} since this event is a non-trivial linear equation on $Y_{i_1}[j_1 - 1]$ and $Y_{i_2}[j_2 - 1]$ and they are independent to each other.

(2) $\Pr[\mathbf{B2}] \leq \sigma^2/2^{n+1}$: This case is similar to the previous case.

(3) $\Pr[\mathbf{B3}] \leq 1/2^{n/2}$: The event **B3** is a multi-collision event for randomly chosen σ many $n/2$-bit strings as Y values are mapped in a regular manner (see the feedback function) to R values. From the union bound, we have

$$\Pr[\mathbf{B3}] \leq \binom{\sigma}{\frac{n}{2}+1} \frac{1}{2^{\frac{n^2}{4}}} < \frac{\sigma^{\frac{n}{2}+1}}{2^{\frac{n^2}{4}}} \leq \left(\frac{\sigma}{2^{(n/2)-1}}\right)^{\frac{n}{2}+1} \leq \frac{1}{2^{n/2}},$$

where the last inequality follows from the assumption $\sigma \leq 2^{(n/2)-2}$ since otherwise the theorem is trivially true.

(4) $\Pr[\mathbf{B4} \wedge \mathbf{B3}^c] \leq nq_f/2^{n/2+1}$: We can assume that **B3** does not hold so the maximum number of multi-collision on R-values is at most n. Now fix (i_1, j_1) with $i_i \neq i'$ and hence due to randomness of L_{i_1} the probability of this case is at most $1/2^{n/2}$. Let us assume that $i_1 = i'$ and so $j_1 \neq p_i + 1$. Once again it is easy to see that $X_i^*[p_i+1] = X_{i'}[j_1]$ reduces to a non-trivial equation in $L_{i'}$. Thus, the probability of this case is also at most $1/2^{n/2}$. By union bound the probability of this event is at most $0.5n/2^{n/2}$ for all i. Summing over all decryption queries, we get $\Pr[\mathbf{B4} \wedge \mathbf{B3}^c] \leq nq_f/2^{n/2+1}$.

(5) $\Pr[\mathbf{B5} \wedge \mathbf{B2}^c] \le q_f/2^{n/2}$: As **B2** does not hold, there can be at most one (i_1, j_1) for which $Y_{i_1}[j_1] = T_i^*$ (for a given i). If there is any such (i_1, j_1), $X_i^*[p_i+1] = X_{i_1}[j_1]$ can hold with probability at most $1/2^{n/2}$. Summing over all decryption queries, we get $\Pr[\mathbf{B5} \wedge \mathbf{B2}^c] \le q_f/2^{n/2}$.

(6) $\Pr[\mathbf{B6}] \le q_f/2^{n/2}$: This is a non-trivial equation in $L_{i'}$ and hence it holds with probability at most $1/2^{n/2}$ for every i. Thus, $\Pr[\mathbf{B6}] \le q_f/2^{n/2}$.

(7) $\Pr[\mathbf{B7} \wedge \mathbf{B3}^c] \le \frac{2q\sigma q_f}{2^{3n/2}}$:
For a fixed i, we have

$$\Pr[X_i^*[p_i+1] = X_{i_1}[0]] = \Pr[(G+I)\cdot Y_{i'}[p_i] \oplus L_{i'}^{p_i} \oplus C_i^*[p_i+1] = N_{i_1}],$$

where $L_{i'}^{p_i}$ is the L value for the p_i-th index of the i'-th encryption query. This is bounded by $1/2^{n/2}$. Now given $L_{i'}$, (the randomness of the first collision), $X_i^*[p_i+2] = (G+I)\cdot Y_{i_1}[0] \oplus L_{i'}^{p_i+1} \oplus C_i^*[p_i+2]$ has $(n-1)$-bit entropy of $(G+I)\cdot Y_{i_1}[0]$ (since $G+I$ has rank $n-1$). So,

$$\Pr[\mathbf{B7} \wedge \mathbf{B3}^c] \le q_f \cdot \frac{q}{2^{n/2}} \cdot \frac{2\sigma}{2^n} = \frac{2q\sigma q_f}{2^{3n/2}}.$$

Summarizing, we have

$$\begin{aligned}
\Pr_{\mathsf{ideal}}[\tau \notin \mathcal{V}_{\text{good}}] &\le \Pr[\mathbf{B1}] + \Pr[\mathbf{B2}] + \Pr[\mathbf{B3}] + \Pr[\mathbf{B4} \wedge \mathbf{B3}^c] + \Pr[\mathbf{B5} \wedge \mathbf{B2}^c] \\
&\quad + \Pr[\mathbf{B6}] + \Pr[\mathbf{B7} \wedge \mathbf{B3}^c] + \Pr[\mathbf{B8}] \\
&\le \frac{1 + \frac{nq_f}{2} + 2q_f}{2^{n/2}} + \frac{\sigma^2}{2^n} + \frac{2q\sigma q_f}{2^{3n/2}} \\
&\le \frac{1}{2^{n/2}} + \frac{q_f(n+4)}{2^{n/2+1}} + \frac{3\sigma^2}{2^n}.
\end{aligned}$$

For the last inequality we assume $q_f \le 2^{n/2}$ and $q \le \sigma$ since otherwise the bound is trivially true. This 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

$$\tau = ((N_i, A_i, M_i, Y_i)_{i\in\{1,\dots,q\}}, (N_{i'}^*, A_{i'}^*, C_{i'}^*, T_{i'}^*, Z_{i'}^*)_{i'\in\{1,\dots,q_f\}}),$$

where $Z_{i'}^* = \perp$. We separate τ into

$$\tau_e = (N_i, A_i, M_i, Y_i)_{i\in\{1,\dots,q\}} \text{ and } \tau_d = (N_{i'}^*, A_{i'}^*, C_{i'}^*, T_{i'}^*, Z_{i'}^*)_{i'\in\{1,\dots,q_f\}},$$

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

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

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 (represented by $X_{i_1}[j_1]$s) in τ_e and $X_i^*[j_2]$ for all $j_2 \neq j$. For $j = p_i + 1$, the above condition is fulfilled by **B4** except the case when $X_{p_1+1}^*[j]$ collides with some nonce in τ_e and it is not the last block. This case, fulfilled by $\mathbf{B5}, \mathbf{B6}$ and **B7** holds for $j = p_i + 2$. Thus, depending on the cases, $X_i^*[p_i + 1]$ or $X_i^*[p_i + 2]$ are fresh and *almost* uniformly random (almost due to 1-bit entropy degradation, since the rank of $G + I$ is $n - 1$). Hence, all the subsequent X^* values are also fresh and almost uniform 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

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

Combining all, we have

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

□

5.2 Brief summary of security analysis of GIFT

The thorough security analysis of GIFT-128 is provided in Section 4 of [4] and by third party cryptanalysis. Here we highlight several important features.

Differential cryptanalysis. Zhu et al. applied the mixed-integer-linear-programming based differential characteristic search method for GIFT-128 and found an 18-round differential characteristic with probability 2^{-109} [25], which was further extended to a 23-round key recovery attack with complexity $(Data, Time, Memory) = (2^{120}, 2^{120}, 2^{80})$. We expect that full (40) rounds are secure against differential cryptanalysis.

Linear cryptanalysis. GIFT-128 has a 9-round linear hull effect of $2^{-45.99}$, which means that we would need around 27 rounds to achieve correlation potentially lower than 2^{-128}. Therefore, we expect that 40-round GIFT-128 is enough to resist against linear cryptanalysis.

Integral attacks. The lightweight 4-bit S-box in GIFT may allow efficient integral attacks. The bit-based division property is evaluated against GIFT-128 by the designers, which detected a 11-round integral distinguisher.

Meet-in-the-middle attacks. Meet-in-the-middle attack exploits the property that a part of key does not appear during a certain number of rounds. The designers and the follow-up work by Sasaki [23] showed the attack against 15-rounds of GIFT-64 and mentioned the difficulty of applying it to GIFT-128 because of the larger ratio of the number of subkey bits to the entire key bits per round; each round uses 32 bits and 64 bits of keys per round in GIFT-64 and GIFT-128, respectively, while the entire key size is 128 bits for both.

6 Hardware Implementation Details

The COFB mode was designed with rate 1, that is every message block is processed only once. Such designs are not only beneficial for throughput, but also energy consumption. However the design does need to maintain an additional 64 bit state, which requires a 64-bit register to additionally included in any hardware circuit that implements it. Although this might not be energy efficient for short messages, in the long run COFB performs excellently with respect to energy consumption. The GIFT block cipher was designed with a motivation for good performance on lightweight platforms. The roundkey additon for the cipher is over only half the state and the keyschedule being only a bit permutation does not require logic gates. These characteristics make the GIFT family of block ciphers well suited for lightweight applications. In fact as reported in [3], among the block ciphers defined for 128-bit block size GIFT-128 has the lowest hardware footprint and very low energy consumption. Thus GIFT-COFB combines the best of both the advantages of the design ideologies.

6.1 Hardware API

NIST has yet to publish a hardware API for the evaluation of the lightweight candidates, and the discussion about the best way forward is still ongoing. Hence we use a minimal API, designed to be simple enough such that it can easily be plugged into existing systems and ensures that any AEAD scheme can be used in all possible configuration such as no associated data or plaintexts blocks and partially filled blocks. Our reasoning for favoring this simpler API is to ensure that no significant energy is consumed to handle the API itself, e.g. the CAESAR HW API [12] requires padding to be done by the circuit, which brings a large array of multiplexers and amplifies the energy consumption for each loaded authenticated data and message block. Nonetheless, a preprocessor circuit could be placed before our AE schemes to ensure CAESAR HW API compatibility. The individual signals are defined in the following way:

CLK, RST: System clock and active-low reset signal. We distinguish two different clock rates; 10 MHz for the partially unrolled versions and 20 MHz for the fully unrolled implementations. Inverse gating technique uses only the first phase of the clock cycle to compute the full block cipher call, therefore the clock period is doubled to ensure all glitches are stabilized during this clock phase.

KEY, NONCE: Key and nonce vectors. These signals are stable once the circuit is reset and are kept active during the entire computation.

DATA: Single data vector that comprises both associated data and regular plaintext material. This choice saves an additional large multiplexer, since all the schemes process associated data and plaintext blocks separately and not in parallel.

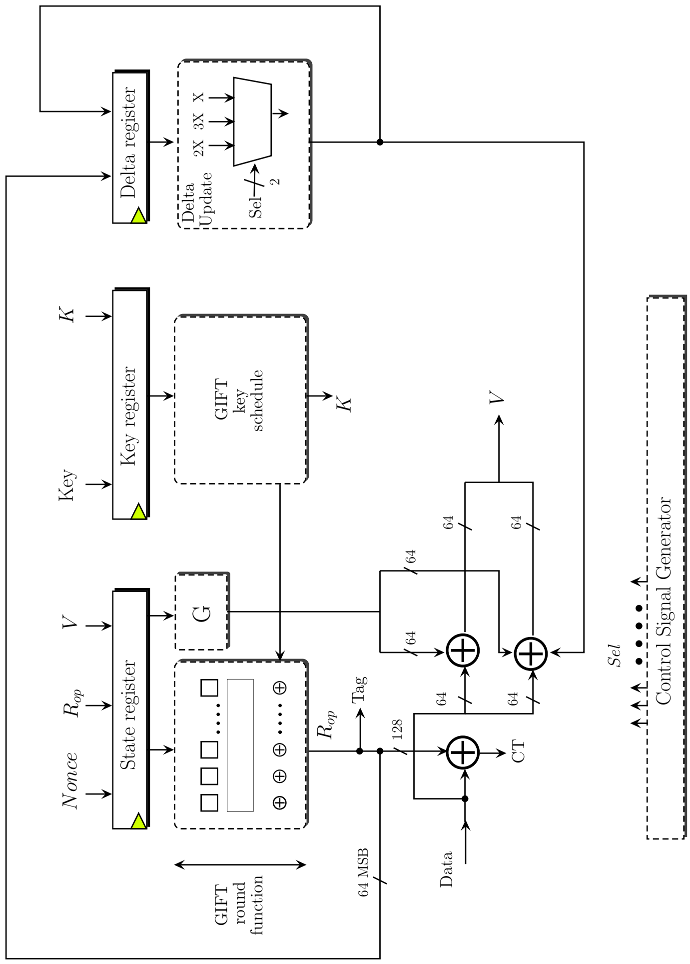

Figure 4: Hardware circuit for round based GIFT-COFB

EAD, EPT: Single bit signals that indicate whether there are no associated data blocks (EAD) or no plaintext blocks (EPT). Both signals are supplied with the reset pulse and remain stable throughout the computation.

LBLK, LPRT: Single bit signals that indicate whether currently processed block is the last associated data block or the last plaintext block (LBLK), and also whether it is partially filled (LPRT). Both signals are supplied alongside each data block and remain stable during its computation.

BRDY, ARDY: Single bit output indicators whether the circuit has finished processing a data block and a new one can be supplied on the following rising clock edge (BRDY) or the entire AEAD computation has been completed (ARDY).

CT, TAG: Separate ciphertext and tag vectors. This again saves an additional multiplexer in schemes where the ciphertext and tag are not ready at the same time, or they appear at different wires.

Figure 4 details the hardware circuit for round based GIFT-COFB. The mode is designed to require one additional 64-bit state apart from the ones used in the block cipher circuit. Thus the design requires an additional 64-bit register. The initial nonce (denoted by $Nonce$ in the above figure) to the encryption routine, and other control signals are generated centrally depending on the length of the plaintext and associated data. Depending on the phase of operation the state register may need to feed either the nonce, the output of the GIFT-128 round function, which is the sum of the encryption output, associated data/plaintext and the additional state *Delta*.

The state *Delta* is updated by multiplying with suitable filed elements of the form $\gamma = \alpha^x(1+\alpha)^y$ with $x + y \leq 4$. Thus we allocate 4 clock cycles to compute the potential Delta update signal. Depending on the value of γ, we update the *Delta* register by either doubling, tripling or the identity operation. For example if $\gamma = \alpha^2$, we execute doubling for 2 cycles and the identity operation for 2 more cycles. Thus in addition to the field operation, the circuit requires a 3:1 multiplexer controlled by a *Sel* signal generated centrally.

6.2 Timing

The GIFT-128 block cipher takes $T_E = 40$ cycles to complete one encryption function. This is the number of clock cycles required in the encryption of the nonce. Each block of associated data would take T_E cycles to process. Before each block of associated data or plaintext is processed we spend $D_u = 4$ cycles to update the *Delta*. Thus if n_a, n_m are the total number of associated data/ message blocks an encryption pass requires $T = T_E + (n_a + n_m)(T_E + D_u)$ cycles to compute.

6.3 Clock Gating

The state register in Figure 4, requires an additional Enable signal to prevent overwrite when the Delta register is being computed. A flip-flop with such an additional functionality usually requires more hardware area. One could circumvent this requirement by gating the clock signal input to the flip-flop bank, so as to prevent unwanted overwrites. This not only brings down the area of the circuit but also power and energy consumptions.

6.4 Performance

We present the synthesis results for the design. The following design flow was used: first the design was implemented in VHDL. Then, a functional verification was first done using Mentor Graphics Modelsim software. The designs were synthesized using the standard cell library of the 90nm logic process of STM (CORE90GPHVT v2.1.a) with the Synopsys Design Compiler, with the compiler being specifically instructed to optimize the circuit for area. A timing simulation was done on the synthesized netlist. The switching activity of each gate of the circuit was collected while running post-synthesis simulation. The average power was obtained using Synopsys Power Compiler, using the back annotated switching activity.

Our smallest implementation of GIFT-COFB (with clock gating) occupied 3271 GE. The power consumed at an operating frequency of 10 MHz is 118.8 μW. The energy consumption figures for various lengths of data inputs are given in the first two rows of Table 2.

6.5 Threshold Implementation

The algebraic degree of the GIFT S-box is 3 (same as PRESENT) and as such constructing threshold circuits is slightly more difficult than for quadratic S-boxes, since it is known that a threshold construction of any function with algebraic degree d requires at least $d+1$ shares [6]. However threshold implementations of the round-based GIFT-128 circuit has been extensively studied in [11]. Since the S-box is cubic, the number of direct shares it must be decomposed to needs to be at least 4. However, the authors in [11] report three philosophies.

Table 2: Implementation results for GIFT-COFB. (Power reported at 10 MHz). Circuits with clock gating are suffixed by "-CG". The notations (xSK), (xS) denote circuits with x shares with/without keypath shared.

Configuration	Clock Gated	Area (GE)	Power (μW)	Energy(nJ) AD 16B	Energy(nJ) PT 32B	Energy(nJ) AD 16B	Energy(nJ) PT 128B	Energy(nJ) AD 16B	Energy(nJ) PT 800B
Unshared									
COFB	NO	3446	122.0	2.098		5.319		27.865	
COFB-CG	YES	3271	118.8	2.043		5.180		27.134	
4 Shares									
COFB(4S)	NO	20292	794.0	13.657		34.618		181.350	
COFB(4S)-CG	YES	19506	789.6	13.581		34.427		180.345	
COFB(4SK)	NO	22510	896.7	15.423		39.096		204.806	
COFB(4SK)-CG	YES	21697	902.0	15.514		39.327		206.017	
3 Shares									
COFB(3S)	NO	11186	423.7	14.067		35.421		184.902	
COFB(3S)-CG	YES	10555	400.6	13.300		33.490		174.822	
COFB(3SK)	NO	13131	504.8	16.759		42.201		220.294	
COFB(3SK)-CG	YES	12179	444.9	14.771		37.194		194.154	

The first decomposes the S-box as the composition $F \circ G$ of two quadratic S-boxes F, G, and implements each decomposed S-box using 3 shares with a register separating the two shared implementations, as in [16]. As such complete evaluation of the substitution layer requires 2 clock cycles instead of one. A second optimization uses the fact that the shares of both G, F are algebraically similar to each other, and differs only in the order of input bits. Hence the authors can further apply an optimization due to [13], that reduces the area of the circuit by implementing the shares over 3 cycles, using a multiplexer to permute the order of bits each time. The third is a direct sharing approach using 4 shares.

For this work, since we focus on energy minimization as an additional optimizable metric, we focus on only the constructions that evaluate the Substitution layer in at most two cycles. Thus we adopted two approaches:

1. **Direct Sharing using 4 shares:** A direct implementation using 4 shares is a straightforward one as the GIFT s-box has an algebraic degree of 3. This circuit requires 4 registers to implement each state share as well as 4 registers to store the shared values of Δ. One may choose or not to share the key path which would require one or 4 registers to implement the keyschedule.

   Since the s-box computation can be cone in one cycle, the number of cycles that this circuit takes to compute the ciphertext/tag pair is the same as the unshared version. Figure 5, gives a block level representation of the circuit (the key path is omitted for simplicity). As can be seen in the figure, depending on whether the key path is shared or not we need 3/6 random 128 bit masks to do all computations.

2. **Decomposing as $F \circ G$ using 3 shares:** Since the GIFT s-box is quadratic, it can be decomposed as $F \circ G$[1], where F and G are quadratic s-boxes. Each of these functions can be constructed using 3 shares. To prevent propagation of glitches from the G to the F layer, we need to put register banks in between them. Hence one substitution layer evaluation is carried out over 2 clock cycles, computation of an

[1] for the exact description of the algebraic expressions for the shared F, G, S boxes please refer to [11]

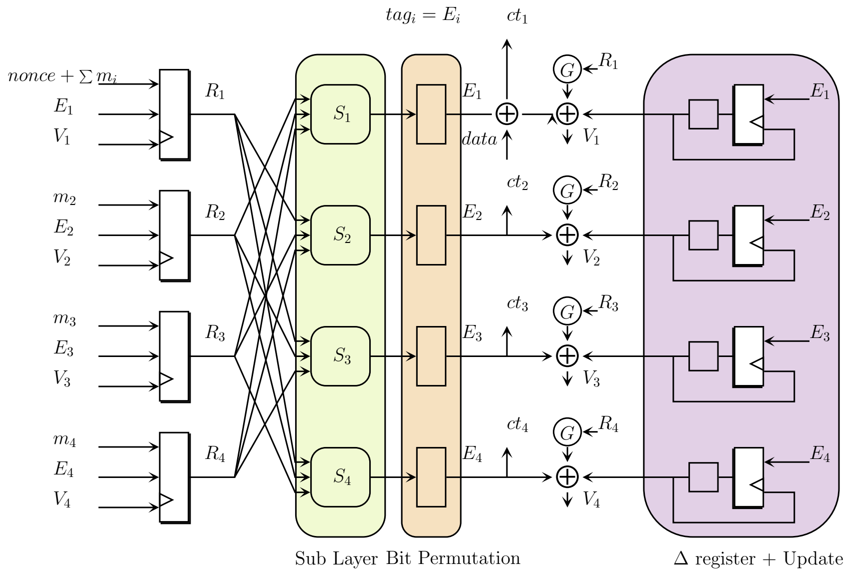

Figure 5: GIFT-COFB using 4 shares (key path is omitted for simplicity)

encryption operation requires $2 \cdot T_E = 80$ cycles. Hence an encryption pass requires $T = 2 \cdot T_E + (n_a + n_m)(2 \cdot T_E + D_u)$ cycles to compute. So this type of construction is considerably slower. On the other hand, from Figure 6, it is clear that depending on whether the key path is shared, the construction requires 2/4 random 128 bit masks.

Table 2 tabulates detailed experimental results of all threshold circuits constructed with 3 as well as 4 shares. The smallest threshold circuit, is the one with 3 shares after applying clock gating and occupies 10555 GE.

7 Software Implementation Details

In this section, we discuss software implementation of GIFT-128. Due to its inherent bitslice structure, it seems natural to consider that the most efficient software implementations of GIFT-128 will be a bitslice strategy, which also offers a constant-time guarantee. This is also the reason why we have used bitslice loading of plaintext/key when using GIFT-128 in the operating mode. The COFB mode being rate-1 and quite simple, as long as a non-parallel implementation is used the entire GIFT-COFB primitive will have similar throughput to GIFT-128 as the input to be handled becomes longer.

Indeed, since COFB is not a parallel operating mode, one can't use several consecutive encryption blocks, which might prevent us to fully use the power of bitslice implementations. More precisely, as the GIFT-128 Sbox size is 4 bits, one will need x parallel blocks on a $32x$-bit architecture. This fits perfectly architecture of 32-bit or less. For bigger registers, one can simply use dummy extra blocks (blocks with random or zero data) to simulate a real bitslice implementation (1 dummy block for 64-bit registers, 3 dummy blocks for 128-bit registers, etc.), which will of course lead to an efficiency penalty. We note however that on a server communicating with several clients, one could consider avoiding the dummy blocks penalty by ciphering all these communications in parallel.

Assume then an architecture with 32-bit registers. The 128-bit plaintext, already in bitslice form, is directly loaded in four registers (similarly for the key). The implementation of the Sbox is straightforward and is provided below. It requires only 6 XORs, 3 ANDs, 1

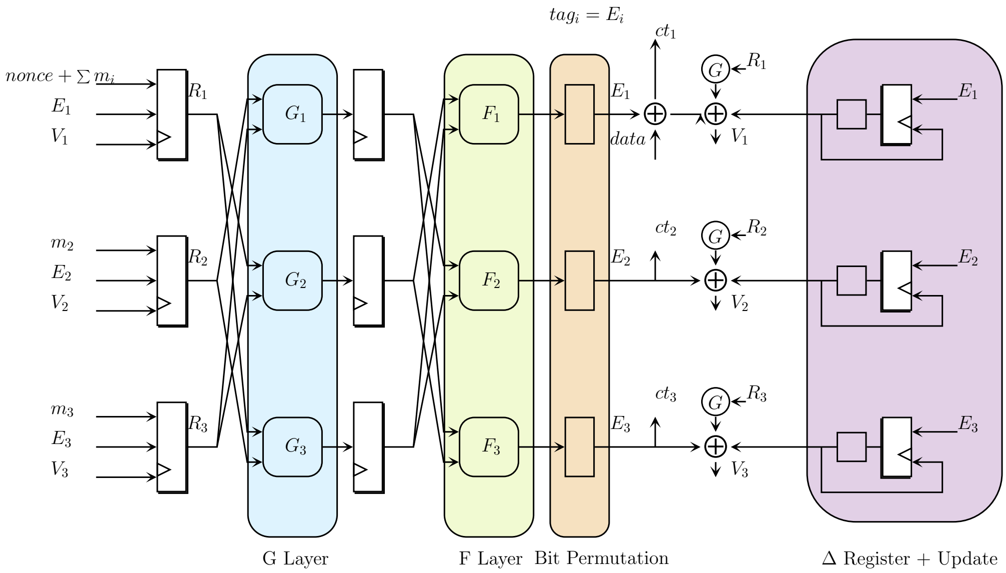

Figure 6: GIFT-COFB using 3 shares (key path is omitted for simplicity)

OR and 1 NOT instruction.

```
/* Input: (MSB) x[3], x[2], x[1], x[0] (LSB) */
x[1] = x[1] XOR (x[0] AND x[2]);
t    = x[0] XOR (x[1] AND x[3]);
x[2] = x[2] XOR (t OR x[1]);
x[0] = x[3] XOR x[2];
x[1] = x[1] XOR x[0];
x[0] = NOT x[0];
x[2] = x[2] XOR (t AND x[1]);
x[3] = t;
/* Output: (MSB) x[3], x[2], x[1], x[0] (LSB) */
```

Figure 7: Software-optimized implementation of the GIFT Sbox.

Applying the subkeys and constants is also straightforward with XOR instructions (one could even consider that subkeys/constants are precomputed and stored in memory). A much more difficult task if to apply the bit permutation, as it is quite costly the move individual bits around in software. A crucial property of the GIFT bit permutations is that a bit in slice i is always sent to the same slice i during this permutation. Thus, applying the bit permutation layer means simply permuting the ordering of the bits inside the registers independently. Fortunately, we have found a new representation of the GIFT-64 and GIFT-128 bit permutations that makes it efficient and simple to implement in software. This strategy, named *fix-slicing* [2], indeed leads to very efficient one-block constant-time GIFT-128 implementations on 32-bit architectures such as ARM Cortex-M family of processors (79 cycles/ byte on ARM Cortex-M3), making GIFT-COFB one of the most efficient candidate according to microcontroller benchmarks [17, 24]. Using smaller architecture will not be an issue as we will actually save more operations comparatively, since part of the bit permutation can be done by proper unrolling and register scheduling. This is confirmed with 8-bit AVR benchmarks [17, 24] where GIFT-COFB is again ranked among the top candidates. Note that using exactly this implementation will also provide decent performance on recent high-end processors (and excellent performances if parallel computations of GIFT-COFB instances are considered and vector instructions are used).

8 Other Implementation/Benchmarking Results on GIFT-COFB

8.1 Software Benchmarking by Renner et. al. [17]

This benchmark results are mainly obtained on five different microcontroller unit platforms. The results are based on the custom made performance evaluation framework, introduced at the NIST LWC Workshop in November 2019. Precisely, the result contains speed, ROM and RAM and benchmarks for software implementations of the 2nd round candidates. We would like to point that, though GIFT-COFB is not designed for microcontrollers, it still stands among the top five designs. The detailed table can be found in [17].

8.2 Software Implementations and Benchmarking by Weatherley [24]

Rhys Weatherley provides efficient 8-bit AVR and 32-bit ARM Cortex-M3 implementations of GIFT-COFB using the fix-slicing strategy. All these implementations are available on the corresponding GitHub repository and benchmarks on these two platforms are provided. Again, we point that, though GIFT-COFB is not designed for microcontrollers, it still ranks at 3rd place among all NIST competition candidates.

8.3 Hardware Benchmarking by Rezvani et. al. [18]

This work implements 6 NIST LWC Round 2 candidates SpoC, GIFT-COFB, COMET-AES, COMET-CHAM, ASCON, and Schwaemm and Esch, on Artix-7, Spartan-6, and Cyclone-V. The results show that SpoC, GIFT-COFB and COMET-CHAM achieves the lowest increase in dynamic power with increasing frequency.

8.4 Hardware Benchmarking by Rezvani et. al. [19]

This work implements three NIST LWC Round 2 candidates GIFT-COFB, SpoC and Spook and few other CAESAR candidates on Artix7. All the implementations are validated on the CAESAR API. The results depict that GIFT-COFB has the highest throughput-to-area (TPA) ratio at 0.154 Mbps/LUT which is a 4.4 factor margin over Spook.

9 Conclusion

In this work, we presented a lightweight and efficient AEAD scheme GIFT-COFB that instantiate AEAD operating mode COFB with block cipher GIFT. In comparison with the previous publications [3, 8], small but significant tweaks are introduced to both COFB and GIFT to further improve the efficiency and performance. With provable security bounds for the operating mode and thorough security analysis, including third party cryptanalysis, on the underlying block cipher primitive, GIFT-COFB is one of the more well-established and competitive candidates in the NIST lightweight cryptography competition.

References

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

[2] Alexandre Adomnicai, Zakaria Najm, and Thomas Peyrin. Fixslicing: A new GIFT representation. *IACR Cryptol. ePrint Arch.*, 2020:412, 2020.

[3] 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 *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 321–345, 2017.

[4] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Siang Meng Sim, Yosuke Todo, and Yu Sasaki. Gift: A small present. Cryptology ePrint Archive, Report 2017/622, 2017. https://eprint.iacr.org/2017/622.

[5] 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.

[6] Begül Bilgin. Threshold implementationsas countermeasure against higher-order differential power analysis. Doctoral Dissertation to K.U.Leuven, 2015. https://www.esat.kuleuven.be/cosic/publications/thesis-256.pdf.

[7] 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.

[8] Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, and Mridul Nandi. Blockcipher-based authenticated encryption: How small can we go? In *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 277–298, 2017.

[9] Shan Chen and John P. Steinberger. Tight security bounds for key-alternating ciphers. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 327–350. Springer, 2014.

[10] Ewan Fleischmann, Christian Forler, and Stefan Lucks. Mcoe: A family of almost foolproof on-line authenticated encryption schemes. In Anne Canteaut, editor, *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, volume 7549 of *Lecture Notes in Computer Science*, pages 196–215. Springer, 2012.

[11] 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.

[12] Ekawat Homsirikamol, William Diehl, Ahmed Ferozpuri, Farnoud Farahmand, Panasayya Yalla, Jens-Peter Kaps, and Kris Gaj. Caesar hardware api. Cryptology ePrint Archive, Report 2016/626, 2016. https://eprint.iacr.org/2016/626.

[13] Sebastian Kutzner, Phuong Ha Nguyen, Axel Poschmann, and Huaxiong Wang. On 3-share threshold implementations for 4-bit s-boxes. In Emmanuel Prouff, editor, *Constructive Side-Channel Analysis and Secure Design - 4th International Workshop, COSADE 2013, Paris, France, March 6-8, 2013, Revised Selected Papers*, volume 7864 of *Lecture Notes in Computer Science*, pages 99–113. Springer, 2013.

[14] NIST. Lightweight cryptography project, 2019.

[15] Jacques Patarin. The "coefficients h" technique. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers*, volume 5381 of *Lecture Notes in Computer Science*, pages 328–345. Springer, 2008.

[16] 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.

[17] Sebastian Renner, Enrico Pozzobon, and Jürgen Mottok. NIST LWC Software Performance Benchmarks on Microcontrollers, 2020.

[18] Behnaz Rezvani, Flora Coleman, Sachin Sachin, and William Diehl. Hardware implementations of NIST lightweight cryptographic candidates: A first look. *IACR Cryptol. ePrint Arch.*, 2019:824, 2019.

[19] Behnaz Rezvani and William Diehl. Hardware Implementations of NIST Lightweight Cryptographic Candidates: A First Look, 2019.

[20] Phillip Rogaway. Authenticated-encryption with associated-data. In *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, pages 98–107, 2002.

[21] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings*, pages 16–31, 2004.

[22] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*, pages 373–390. Springer, 2006.

[23] Yu Sasaki. Integer linear programming for three-subset meet-in-the-middle attacks: Application to gift. In Atsuo Inomata and Kan Yasuda, editors, *Advances in Information and Computer Security*, pages 227–243, Cham, 2018. Springer International Publishing.

[24] Rhys Weatherley. Lightweight Cryptography Primitives, 2020.

[25] Baoyu Zhu, Xiaoyang Dong, and Hongbo Yu. Milp-based differential attack on round-reduced gift. Cryptology ePrint Archive, Report 2018/390, 2018. https://eprint.iacr.org/2018/390.