GIFT-COFB

v1.1

Designers/Submitters:	**E-mails:**
Subhadeep Banik	subhadeep.banik@epfl.ch
Avik Chakraborti	avikchkrbrti@gmail.com
Tetsu Iwata	tetsu.iwata@nagoya-u.jp
Kazuhiko Minematsu	k-minematsu@nec.com
Mridul Nandi	mridul.nandi@gmail.com
Thomas Peyrin	thomas.peyrin@ntu.edu.sg
Yu Sasaki	yu.sasaki.sk@hco.ntt.co.jp
Siang Meng Sim	crypto.s.m.sim@gmail.com
Yosuke Todo	yosuke.todo.xt@hco.ntt.co.jp

May 17, 2021

Chapter 1

Introduction

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

This document describes GIFT-COFB authenticated, which instantiates the COFB (COmbined FeedBack) block cipher based AEAD mode with the GIFT block cipher. COFB primarily focuses on the hardware implementation size. Here, we consider the overhead in size, thus the state memory size beyond the underlying block cipher itself (including the key schedule) is the criteria we want to minimize, which is particularly relevant for hardware implementation.

An initial version of COFB was presented in [12] and this latest version of COFB is a minor modification over the original COFB mode.

This version supports all the desirable properties mentioned in the NIST lightweight cryptography portfolio [26], and it is efficient for lightweight implementations as well.

There are many approaches of designing a secure and lightweight block cipher based AEAD. We focus on using a lightweight, well analyzed block cipher and minimizing the total encryption/decryption state size. We deploy a hardware optimized block cipher GIFT-128 [5]. In addition to that, we use combined feedback over the block cipher output and the data blocks along with a tweak dependent secret masking (as used in XEX [30]). This combination helps us to minimize the amount of masking by a factor of 2 from [30].

The COFB mode achieves several interesting features. It achieves a high value for rate which is 1 (i.e, needs only one block cipher call for one input block). The mode is inverse-free, i.e., it does not need a block cipher inverse during decryption. In addition to these features, this mode has a quite small state size, namely $1.5n + k$ bits, in case the underlying block cipher has an n-bit block and k-bit keys.

Chapter 2

Specification

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 $\epsilon \in \mathcal{B}^<$.
- 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 $\{a, a+1, \ldots, b\}$.

2.1.1 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 $(s - 1)$.

2.1.2 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.

64-bit String	Polynomial
$0^{62}10$	α
$0^{62}11$	$\alpha + 1$
$0^{61}100$	α^2

Table 2.1: Various representations of some elements in $\mathbb{F}_{2^{64}}$

Thus, 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 α_{64}. Multiplying an element $b := b_{63}b_{62} \cdots b_0 \in \mathbb{F}_{2^{64}}$ by the primitive element α_{64} of $\mathbb{F}_{2^{64}}$ can be done very efficiently as follows:

$$b \cdot \alpha_{64} = \begin{cases} b \ll 1, & \text{if } a_{63} = 0 \\ (b \ll 1) \oplus 0^{59}11011, & \text{else,} \end{cases}$$

where $\ll r$ denotes left shift by r bits. Throughout this document, we use α to denote α_{64}. 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.

2.2 Recommended Parameter Choice

We propose a construction GIFT-COFB with the underlying block cipher as the only parameter. The block cipher can be chosen by the following recommendation.

1. n: Length of the block cipher state in bits. The recommended choice is $n = 128$.

2. τ: Length of the tag in bits. The recommended choice is $\tau = 128$.

3. E_K: The recommended choice of E_K is the block cipher GIFT-128.

2.3 Input and Output Data

To encrypt a message M with associated data A and nonce N, one needs to provide the information given below.

The encryption algorithm takes as input

- An encryption key $K \in \{0,1\}^{128}$.
- A nonce $N \in \{0,1\}^{128}$. This can include the counter to make the nonce non-repeating.
- Associated data and message $A, M \in \{0,1\}^*$.

It generates the following output data:

- Ciphertext $C \in \{0,1\}^{|M|}$.
- Tag $T \in \{0,1\}^{128}$

To decrypt (with verification) a ciphertext-tag pair (C, T) with associated data A and nonce N, one needs to provide the information given below.

- An encryption key $K \in \{0,1\}^{128}$.
- A nonce $N \in \{0,1\}^{128}$.
- Associated data and ciphertext $A, C \in \{0,1\}^*$.
- Tag $T \in \{0,1\}^{128}$

It generates the following output data:

- Message $M \in \{0,1\}^{|C|} \cup \{\perp\}$, where $\perp$ is a special symbol denoting rejection.

2.4 Mathematical Components

2.4.1 Block cipher GIFT-128

`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. There are two versions of `GIFT-128`, namely GIFT-64 and `GIFT-128`. But since we are focusing only on `GIFT-128` in this document,

we use GIFT-128 and GIFT-128 interchangeably. For the rest of this document, we take the full version of GIFT-128 paper [6] as reference.

There are different ways to perceive GIFT-128, the more pictorial description is detailed in Section 2 of [6], which looks like a larger version of PRESENT cipher with 32 4-bit S-boxes and an 128-bit bit permutation (see Figure 2.1). In this document, we will be using bitslice description which is similar to Appendix A of [6].

Round function

Each round of GIFT-128 consists of 3 steps: SubCells, PermBits, and AddRoundKey.

Initialization. The 128-bit plaintext is loaded into the cipher state S which will be expressed as 4 32-bit segments. In the perspective of a 2-dimensional array, the bit ordering is from top-down, then right to left.

$$S = \begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{bmatrix} \leftarrow \begin{bmatrix} b_{124} & \cdots & b_8 & b_4 & b_0 \\ b_{125} & \cdots & b_9 & b_5 & b_1 \\ b_{126} & \cdots & b_{10} & b_6 & b_2 \\ b_{127} & \cdots & b_{11} & b_7 & b_3 \end{bmatrix}.$$

The 128-bit secret key is loaded into the key state KS partitioned into 8 16-bit words. In the perspective of a 2-dimensional array, the bit ordering is from right to left, then bottom-up.

$$KS = \begin{bmatrix} W_0 & \| & W_1 \\ W_2 & \| & W_3 \\ W_4 & \| & W_5 \\ W_6 & \| & W_7 \end{bmatrix} \leftarrow \begin{bmatrix} b_{127} & \cdots & b_{112} & \| & b_{111} & \cdots & b_{98} & b_{97} & b_{96} \\ b_{95} & \cdots & b_{80} & \| & b_{79} & \cdots & b_{66} & b_{65} & b_{64} \\ b_{63} & \cdots & b_{48} & \| & b_{47} & \cdots & b_{34} & b_{33} & b_{32} \\ b_{31} & \cdots & b_{16} & \| & b_{15} & \cdots & b_2 & b_1 & b_0 \end{bmatrix}$$

Refer to Section 2.4.2 for details of the arriving data.

SubCells. Update the cipher state with the following 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 $\&, \mid$ and $\sim$ are AND, OR and NOT operation respectively.

PermBits. Different 32-bit bit permutations are applied to each S_i independently.

Table 2.2: Specifications of `GIFT-128` bit permutation.

Index	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
S_0	29	25	21	17	13	9	5	1	30	26	22	18	14	10	6	2
S_1	30	26	22	18	14	10	6	2	31	27	23	19	15	11	7	3
S_2	31	27	23	19	15	11	7	3	28	24	20	16	12	8	4	0
S_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
S_0	31	27	23	19	15	11	7	3	28	24	20	16	12	8	4	0
S_1	28	24	20	16	12	8	4	0	29	25	21	17	13	9	5	1
S_2	29	25	21	17	13	9	5	1	30	26	22	18	14	10	6	2
S_3	30	26	22	18	14	10	6	2	31	27	23	19	15	11	7	3

In Table 2.2, 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, bit 1 (the 2nd rightmost bit) of S_1 is shifted 1 position to the right, to the initial position of bit 0, while bit 0 is shifted 8 positions to the left.

AddRoundKey. This step consists of adding the round key and round constant. Two 32-bit segments U, V are extracted from the key state as the round key.

$$RK = U \| V.$$

For the addition of round key, U and V are XORed to S_2 and S_1 of the cipher state respectively.

$$S_2 \leftarrow S_2 \oplus U,$$
$$S_1 \leftarrow S_1 \oplus V.$$

For the addition of round constant, S_3 is updated as follows,

$$S_3 \leftarrow S_3 \oplus \texttt{0x800000XY},$$

where the byte $\texttt{XY} = 00c_5c_4c_3c_2c_1c_0$.

Key schedule and round constants

A round key is *first* extracted from the key state before the key state update. Four 16-bit words of the key state are extracted as the round key $RK = U \| V$.

$$U \leftarrow W_2 \| W_3,\ V \leftarrow W_6 \| W_7.$$

The key state is then updated as follows,

$$\begin{bmatrix} W_0 & \| & W_1 \\ W_2 & \| & W_3 \\ W_4 & \| & W_5 \\ W_6 & \| & W_7 \end{bmatrix} \leftarrow \begin{bmatrix} W_6 \ggg 2 & \| & W_7 \ggg 12 \\ W_0 & \| & W_1 \\ W_2 & \| & W_3 \\ W_4 & \| & W_5 \end{bmatrix},$$

where $\ggg i$ is an i bits right rotation within a 16-bit word.

The round constants are generated using the a 6-bit affine LFSR, whose state is denoted as $c_5c_4c_3c_2c_1c_0$. Its update function is defined as:

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

The six bits 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`

Decryption of `GIFT-128`

We omit the description of the inverse of `GIFT-128` as it is not required for GIFT-COFB.

2.4.2 Format of Incoming Data

As seen in the "Initialization" phase, the loading of the data (plaintext) bits is column-wise. Typically, that would require additional instructions to rearrange and pack the incoming data into the S_i's, and unpack them back to the initial data format after the encryption. Such practice, however, is merely a matter of perspective and does not affect the security. In fact, it costs additional clock cycles in software implementation to pack them into the desired format. To save on this unnecessary overhead, we regard the incoming data and key as having the desired format and load them into the states in the most natural manner.

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

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

where B_i are the arriving bytes.

Relation to `GIFT-128` LUT based implementation

An alternative implementation of `GIFT-128` is using look-up table (LUT) for the SubCells operation. Such implementation prefers having the data in the conventional format, i.e. $B_0B_1 \cdots B_{15} = b_{127}b_{126} \cdots b_1b_0$.

The conversion from an LUT implementation to our bitslice implementation is simple: Note that we perceive the incoming data as bitslice format,

$$\begin{bmatrix} B_0 & \| & B_1 & \| & B_2 & \| & B_3 \\ B_4 & \| & B_5 & \| & B_6 & \| & B_7 \\ B_8 & \| & B_9 & \| & B_{10} & \| & B_{11} \\ B_{12} & \| & B_{13} & \| & B_{14} & \| & B_{15} \end{bmatrix}$$

$$= \begin{bmatrix} b_{124}b_{120}b_{116}\cdots b_{96} & \| & b_{92}\cdots b_{64} & \| & b_{60}\cdots b_{32} & \| & b_{28}\cdots b_0 \\ b_{125}b_{121}b_{117}\cdots b_{97} & \| & b_{93}\cdots b_{65} & \| & b_{61}\cdots b_{33} & \| & b_{29}\cdots b_1 \\ b_{126}b_{122}b_{118}\cdots b_{98} & \| & b_{94}\cdots b_{66} & \| & b_{62}\cdots b_{34} & \| & b_{30}\cdots b_2 \\ b_{127}b_{123}b_{119}\cdots b_{99} & \| & b_{95}\cdots b_{67} & \| & b_{63}\cdots b_{35} & \| & b_{31}\cdots b_3 \end{bmatrix}.$$

First, unpack the data into the conventional format. Next, perform the LUT implementation of `GIFT-128`. Finally, pack the output data back to the bitslice format. No additional packing/unpacking is required for the key. This yields the exact same bitslice implementation as we described in the Section 2.4.1.

Test Vectors

Key	:	`00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F`
Plaintext	:	`00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F`
Ciphertext	:	`A9 4A F7 F9 BA 18 1D F9 B2 B0 0E B7 DB FA 93 DF`
Key	:	`E0 84 1F 8F B9 07 83 13 6A A8 B7 F1 92 F5 C4 74`
Plaintext	:	`E4 91 C6 65 52 20 31 CF 03 3B F7 1B 99 89 EC B3`
Ciphertext	:	`33 31 EF C3 A6 60 4F 95 99 ED 42 B7 DB C0 2A 38`

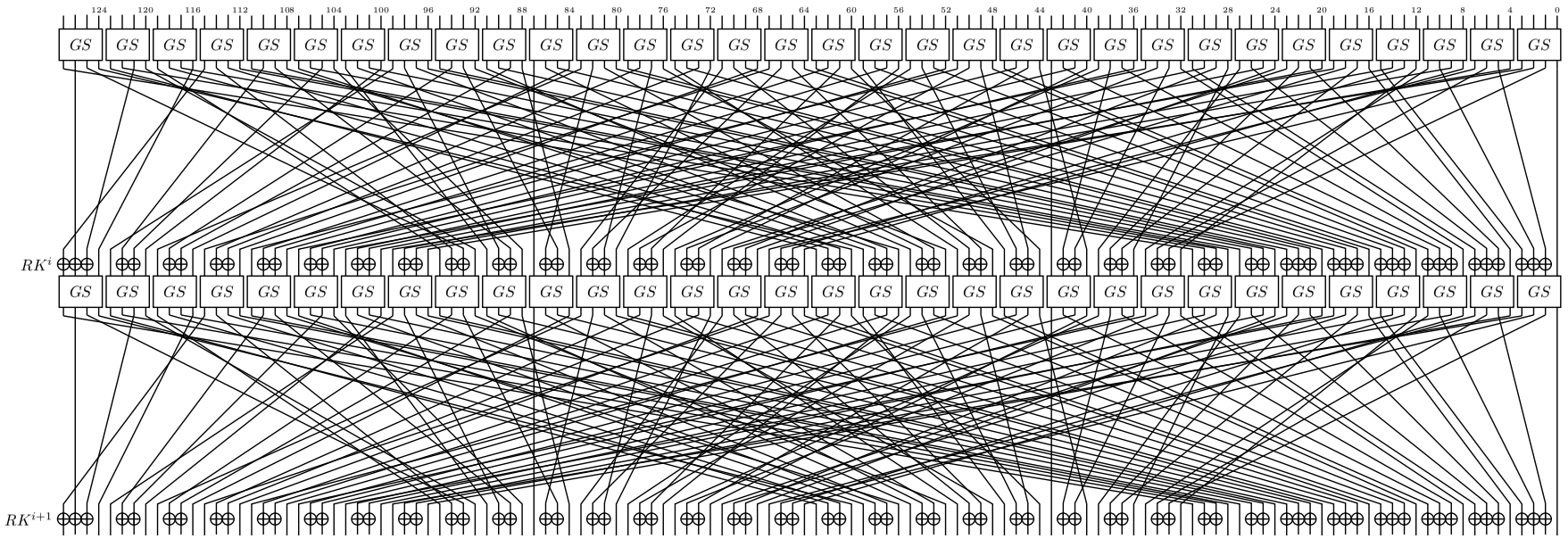

Figure 2.1: 2 rounds of `GIFT-128`.

2.5 COFB Authenticated Encryption Mode

In this section, we present our proposed mode, COFB in Fig. 2.3. We first specify the basic building blocks and parameters used in our construction.

Key and Block cipher. The underlying cryptographic primitive is an n-bit block cipher, E_K. We assume that n is a multiple of 4. The key of the scheme is the key of the block cipher, i.e. K.

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

$$\mathsf{Pad}(x) = \begin{cases} x & \text{if } x \neq \epsilon \text{ and } |x| \bmod n = 0 \\ x \| 10^{(n-(|x| \bmod n)-1)} & \text{otherwise.} \end{cases} \tag{2.1}$$

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

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

Note that when $(X, M) = \rho'(Y, C)$ then $X = (G \oplus I) \cdot Y \oplus C$, where I is the $n \times n$ identity matrix. Our choice of G ensures that $G \oplus I$ has rank $n-1$. When Y is chosen randomly for both computations of X (through ρ and ρ'), X also behaves randomly. We need this property when we bound probability of bad events later.

We present the specifications of COFB in Fig. 2.3, where α and $(1+\alpha)$ are written as 2 and 3. See also Fig. 2.2. The encryption and decryption algorithms are denoted by COFB-$\mathcal{E}_K$ and COFB-$\mathcal{D}_K$. We remark that the nonce length is n 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 $|T| = n$. The decryption algorithm takes (N, A, C, T) and outputs M or $\perp$.

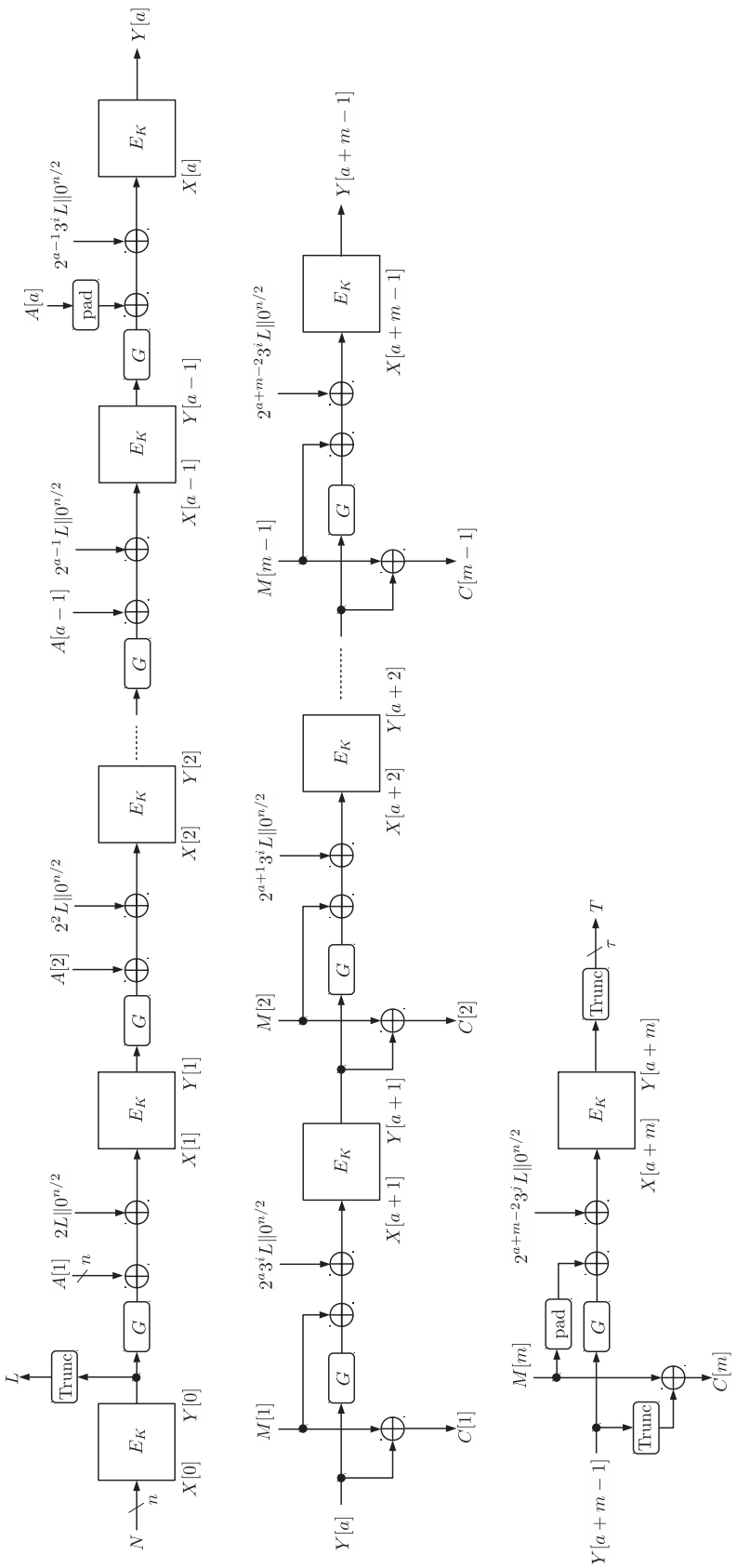

Figure 2.2: Encryption of COFB. In the rightmost figure, the case of encryption for empty M (hence a MAC for (N, A)) can be highlighted as $T = \mathsf{Trunc}_\tau(Y[a])$

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

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

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

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

Figure 2.3: The encryption and decryption algorithms of COFB

Chapter 3

Performance

3.1 Hardware Performance

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-128` block cipher was designed with a motivation for good performance on lightweight platforms. The round key addition for the cipher is over only half the state and the key schedule being only a bit permutation does not require logic gates. These characteristics make the `GIFT-128` well suited for lightweight applications. In fact as reported in [5], 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.

Figure 3.1 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

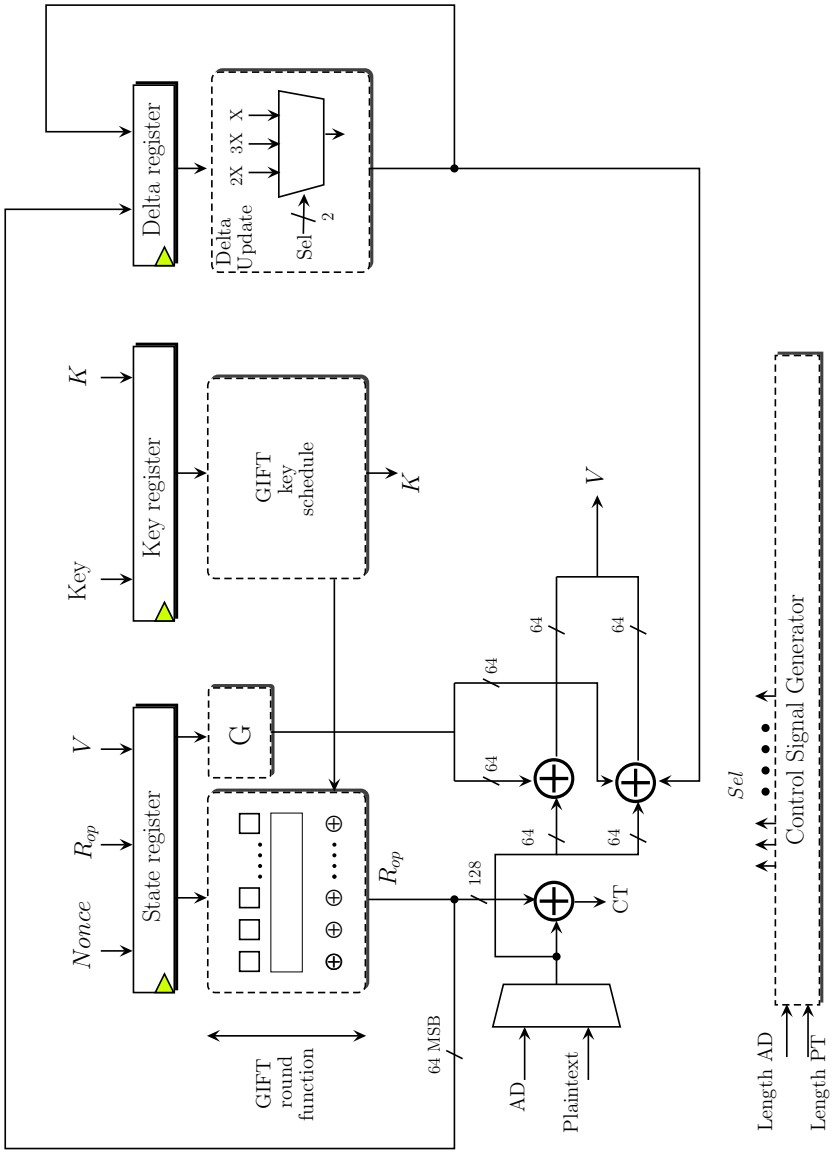

Figure 3.1: Hardware circuit for round based GIFT-COFB

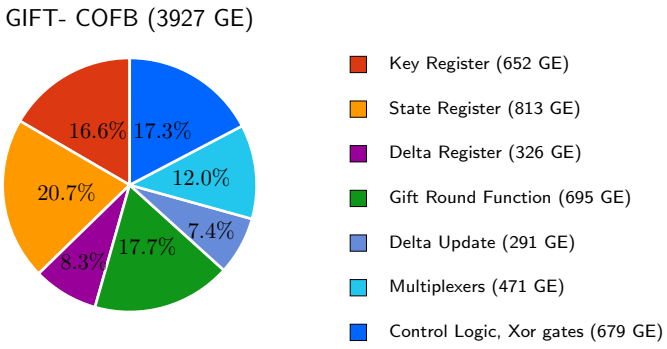

Figure 3.2: Component-wise breakup of the GIFT-COFB circuit

more cycles. Thus in addition to the field operation, the circuit requires a 3:1 multiplexer controlled by a *Sel* signal generated centrally.

3.1.1 Timing

The `GIFT-128` block cipher takes $E = 40$ 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 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 $T = E + (n_a + n_m)(E + D_u)$ cycles to compute.

3.1.2 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 implementation of GIFT-COFB occupied 3927 GE. A component-wise breakup of the circuit is given in Figure 3.2. The power consumed at an operating frequency is 156.3 μW. The energy consumption figures for various lengths of data inputs are given in Table 3.1.

Block Cipher	Area (GE)	Power(μW)	Energy(nJ)					
			AD	PT	AD	PT	AD	PT
			0B	16B	16B	16B	16B	32B
GIFT-128	3927	156.3	1.31		2.00		2.69	

Table 3.1: Implementation results for GIFT-COFB. (Power reported at 10 MHz)

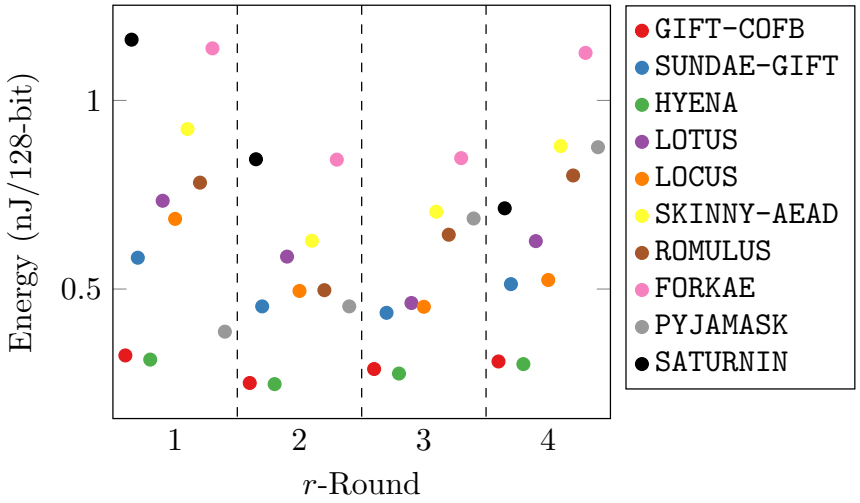

Figure 3.3: Energy consumption (nJ/128-bit) comparison chart for the r-round partially-unrolled implementations with $r \in \{1, 2, 3, 4\}$. For each candidate the best obtained energy value obtained through techniques is used.

3.1.3 Energy Efficiency

Some more results were recently published in [9] that compares the the energy efficiency of GIFT-COFB with 9 other modes of operation using lightweight block ciphers in the NIST LWC. Because of the excellent energy characteristics of GIFT-128 and the fact that GIFT-COFB is a rate 1 mode, GIFT-COFB was found to be one of the most energy efficient designs in the NIST LWC. We experimented with different round unrolled architectures of the core block cipher used in the design (from round-based to fully unrolled) using the TSMC 90nm standard cell library. Figure 3.3 charts the optimal energy per 128-bit block value for each degree of unrolling r and candidate. Table 3.2 details the simulation results. Note that all the charts and tables are taken from [9].

3.1.4 Threshold Implementations

The s-box of GIFT-128 belongs to the cubic class $\mathcal{C}_{172}$ which is decomposable into 2 quadratics. The algebraic expressions of the output shares of both the 3 and

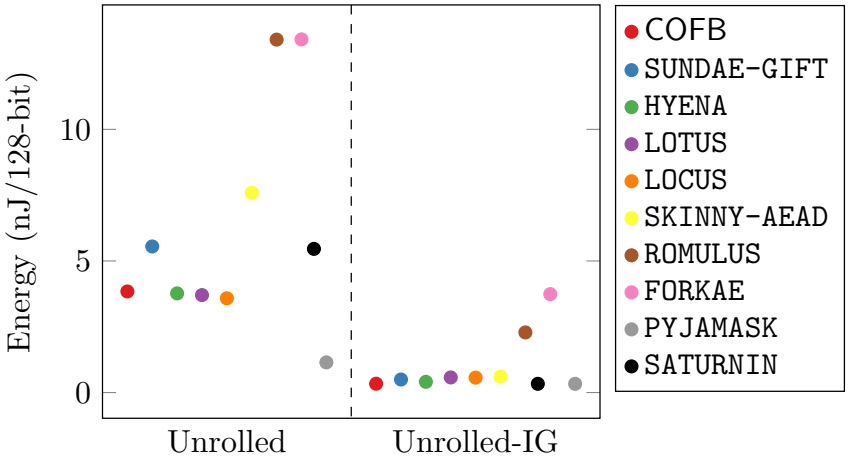

Figure 3.4: Energy consumption (nJ/128-bit) comparison chart for the fully-unrolled implementations with and without inverse-gating.

4-share TI can be found in [16]. Table 3.3 lists the simulation results using the same measurement setup as the unshared round-based implementations. It can be seen that GIFT-COFB offers both low area and competitive energy efficiency when compared with other modes of operation.

3.2 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

Table 3.2: Various GIFT-COFB implementations. Latency and energy is given for processing a single authenticated data block followed by eight message blocks.CG denotes clock gated. IG denotes "inverse-gated" implementation as per the generic energy reduction technique explained in [3]

Candidate	Implementation	Latency (cycles)	Area (GE)	TP_{max} (Mbps)	Power (μW)	Energy (nJ/128-bit)
GIFT-COFB	1-Round	400	4710	615.38	69.3	0.363
	1-Round-CG	400	4700	569.17	61.9	0.324
	2-Round	200	5548	1192.55	106.8	0.280
	2-Round-CG	200	5510	952.06	95.5	**0.251**
	3-Round	140	6372	1211.87	159.0	0.293
	3-Round-CG	140	6311	1172.16	156.2	0.288
	4-Round	100	7144	1304.64	237.0	0.314
	4-Round-CG	100	7036	1140.59	232.4	0.308
	Unrolled	10	35735	2015.75	12628.4	3.841
	Unrolled-IG	10	43584	711.15	1107.0	0.337

requires only 6 XORs, 3 ANDs, 1 OR and 1 NOT instruction.

```
1  /* Input: (MSB) x[3], x[2], x[1], x[0] (LSB) */
2  x[1] = x[1] XOR (x[0] AND x[2]);
3  t    = x[0] XOR (x[1] AND x[3]);
4  x[2] = x[2] XOR (t OR x[1]);
5  x[0] = x[3] XOR x[2];
6  x[1] = x[1] XOR x[0];
7  x[0] = NOT x[0];
8  x[2] = x[2] XOR (t AND x[1]);
9  x[3] = t;
10 /* Output: (MSB) x[3], x[2], x[1], x[0] (LSB) */
```

Figure 3.5: Software-optimized implementation of the GIFT Sbox.

Applying the round keys and constants is also straightforward with XOR instructions (one could even consider that round keys/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

Table 3.3: Measurements for the 1-round threshold implementations. The schemes using GIFT-128 are colored in light gray whereas, SKINNY-AEAD based schemes are in white. Note that the table has been taken from [9]

Candidate	Conf.	Shares #	Latency (cycles)	Area (GE)	TP_{max} (Mbps)	Power (mW)	Energy (nJ/128-bit)
GIFT-COFB	CG-RB	3	800	16386	208.9	0.214	2.243
	CG-RB	4	400	25850	350.8	0.358	**1.875**
SUNDAE-GIFT	RB	3	1440	13297	145.7	0.215	3.719
	RB	4	720	21848	285.2	0.357	**2.999**
HYENA	CG-RB	3	800	14769	344.9	0.212	2.216
	CG-RB	4	400	24540	497.4	0.358	**1.875**
LOTUS	CG	3	2072	14176	121.7	0.145	3.581
	CG	4	1036	19712	133.0	0.262	**3.232**
LOCUS	CG	3	2072	12366	121.7	0.137	3.362
	CG	4	1036	17597	176.8	0.255	**3.148**
SKINNY-AEAD	CG	3	2240	18501	92.83	0.2264	6.134
ROMULUS	CG-RB	3	2056	13450	130.00	0.1865	4.656
FORKAE	CG	3	3008	17008	76.60	0.2483	8.304
PYJAMASK	CG-RB	3	348	42001	620.2	0.472	1.825
	CG-RB	4	180	64577	927.6	0.814	**1.628**

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 [27, 32]. 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 [27, 32] 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).

3.3 Other Implementation/Benchmarking Results on GIFT-COFB

3.3.1 Software Benchmarking by Renner et. al. [27]

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 [27].

3.3.2 Software Implementations and Benchmarking by Weatherley et. al. [32]

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.

3.3.3 Hardware Benchmarking by Rezvani et. al. [28]

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.

3.3.4 Hardware Benchmarking by Rezvani et. al. [29]

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.

Chapter 4

Security of GIFT-COFB

Our security claims are summarized in Table 4.1.

Construction	State Size(bits)	IND-CPA(bits)	INT-CTXT(bits)
GIFT-COFB	192 (excluding the key state)	64	58

Table 4.1: IND-CPA and INT-CTXT security of GIFT-COFB under the nonce respecting scenario

In this writeup, we provide a sketch of the security proof for GIFT-COFB. The complete AE security proof of the COFB mode can be found in [4], where the AE advantage of COFB is upper bounded by

$$\mathbf{Adv}^{\mathrm{AE}}_{\mathsf{COFB}}((q,q_f),(\sigma,\sigma_f),t) \leq \mathbf{Adv}^{\mathrm{prp}}_{\mathsf{GIFT}}(q',t') + \frac{\binom{q'}{2}}{2^n} + \frac{1}{2^{n/2}} + \frac{q_f(n+4)}{2^{n/2+1}} + \frac{3\sigma^2 + q_f + 2(q+\sigma+\sigma_f)\cdot\sigma_f}{2^n} \tag{4.1}$$

with $q' = q + q_f + \sigma + \sigma_f$, which corresponds to the total number of block cipher calls through the game, and $t' = t + O(q')$. Note that, the advantage has been taken by the maximum advantage over all the adversaries making q encryption queries, q_f decryption queries and running in time t, such that σ, σ_f are the total number of blocks queried in the encryption and decryption queries respectively. For GIFT-COFB, $n = 128$ and

- GIFT-COFB satisfies the requirement of security against 2^{112} computations in a single key setting. This comes from the security assumption of GIFT. A sketch of the security analysis of GIFT is provided in Section 4.3.
- GIFT-COFB satisfies the requirement that the input size shall not be smaller than $2^{50} - 1$ bytes under a single key. This comes from the above expression

of AE security advantage of COFB (which depicts even a higher bound than $2^{50} - 1$, when $n = 128$). A sketch of the security analysis is provided in Section 4.1 and 4.2.

We would also like to mention that, the upper bound of the input length in blocks for a single query is bounded by 2^{51}. This comes from the fact that for all $(a, b) \in \{0, \ldots, 2^{51}\} \times \{0, \ldots, 10\}$, $2^a 3^b$ (needed to update L) are distinct. This result has been presented by Rogaway in [30].

4.1 IND-CPA Security of GIFT-COFB

To attack against the privacy of GIFT-COFB, we assume that an adversary runs in time t and makes at most q_e encryption queries $(N_i, A_i, M_i)_{i=1\ldots q_e}$ to GIFT-COFB with an aggregate of total σ_e many blocks. In return the adversary receives $(C_i, T_i)_{i=1\ldots q_e}$. In this interaction, the adversary tries to distinguish the construction from a random function with the same domain and range.

If we use a hybrid argument, then we first make a transition by using an n-bit (uniform) random permutation P instead of the underlying block cipher GIFT_K, and then to use an n-bit (uniform) random function R instead of P. This two-step transition requires the first two terms of our bound, from the standard PRP-PRF switching lemma and from the computation to the information security reduction (e.g., see [18]). Then what we need is a bound for COFB using R, denoted by COFB[R].

The adversary can distinguish COFB[R] construction from a random function with the same domain and range if it finds a state collision among the internal states (block cipher inputs) of two encryption queries. It is easy to see that the probability of a collision is bounded by $\frac{\binom{\sigma_e}{2}}{2^{128}}$. This holds as for any two of the σ_e block cipher inputs (corresponding to σ_e input data blocks) are equal with the probability 2^{-128} (from the randomness of the previous block cipher outputs). Hence, the privacy or IND-CPA advantage of GIFT-COFB can be bounded by $\mathbf{Adv}^{\mathrm{prp}}_{\mathrm{GIFT}}(q_e, t) + \frac{\binom{q_e}{2}}{2^{128}} + \frac{\binom{\sigma_e}{2}}{2^{128}}$.

4.2 IND-CTXT Security of GIFT-COFB

On the other hand, to attack against the integrity of GIFT-COFB, assume that an adversary makes at most q_e encryption queries $(N_i, A_i, M_i)_{i=1\ldots q_e}$ to $\mathsf{COFB} - \mathcal{E}_K$ with an aggregate of total σ_e many blocks. In return the adversary receives $(C_i, T_i)_{i=1\ldots q_e}$. The adversary also tries to forge with q_f decryption queries $(N_j^*, A_j^*, C_j^*, T_j^*)_{j=1\ldots q_f}$ with a total number of σ_f blocks to $\mathsf{COFB} - \mathcal{D}_K$ and receives $M_j^* or \perp$. Let $q = q_e + q_f + \sigma_e + \sigma_f$. The trivial solution for forging is to guess the tag which can be bounded by $\frac{q_f}{2^{128}}$ (One of the q_f forged tags is valid).

A bad case **B1** occurs if an adversary can obtain an intermediate block cipher input state collision between an encryption query and a decryption query or

between two decryption queries. The probability of this event is bounded by $\frac{(q_e+\sigma_e+2\sigma_f)\sigma_f}{2^{128}} + \frac{64q_f}{2^{64}}$ (Actually the last term is $\frac{0.5nq_f}{2^{64}}$ and here $n = 128$).

To bound the probability of **B1**, we assume the following bad events do not hold. The bad events are as follows.

- **B2**: Multicollision of size more than $n/2$ (with $n = 128$) on the right half of the intermediate block cipher inputs for the encryption queries do not occur. This event is bounded by a negligible probability $\frac{2\sigma_e}{2^{64}}$.
- **B3**: Let $X_i[j]$ be the j^{th} block cipher input in the i^{th} encryption query and $X_i^*[j]$ be the j^{th} block cipher input in the i^{th} decryption query. For each of the decryption queries, after the prefix p_i (defined in footnote[1]), we define the following event **B3** as

$$X_i^*[p_i + 1] = N_j \text{ and } X_i^*[p_i + 2] = X_{i'}[j'], \text{ for some } i, j, i', j'.$$

  This event is bounded by a negligible probability $\frac{q_f}{2^{64}} + \frac{q_f(q_e-1)}{2^{128}}$.

The part $\frac{(q_e+\sigma_e+2\sigma_f)\sigma_f}{2^{128}}$ in **B1** occurs for block cipher input state collision between an encryption and an decryption query or between two decryption queries. Here, the number of such bad pairs is bounded by $(q_e + \sigma_e + 2\sigma_f)\sigma_f$.

The part $\frac{64q_f}{2^{64}}$ in **B1** occurs for block cipher input state collision between an encryption query and a decryption query. The probability bound comes from the fact that for the i^{th} decryption query, the $(p_i + 1)^{st}$ block cipher input is fresh with high probability due to fact that **B2, B3** do not hold. The $(p_i + 1)^{st}$ block cipher input is not fresh with probability bounded by $\frac{64}{2^{64}}$. It comes from the fact that no multi-collision of size more than 64 occurs.

Forging event should imply one of the bad events. Hence the INT-CTXT advantage of GIFT-COFB is bounded by

$$\frac{q_f}{2^{128}} + \frac{2\sigma_e}{2^{64}} + \frac{q_f}{2^{64}} + \frac{q_f(q_e-1)}{2^{128}} + \frac{(q_e+\sigma_e+2\sigma_f)\sigma_f}{2^{128}} + \frac{64q_f}{2^{64}}$$

$$= \frac{3\sigma_e + q_f}{2^{64}} + \frac{(q_e+\sigma_e+2\sigma_f)\sigma_f + q_f + (q_e-1)q_f}{2^{128}} + \frac{64q_f}{2^{64}}.$$

4.3 Security Analysis of GIFT-128 (Extract)

The security analysis of GIFT-128 is provided in Section 4 of [6]. Here we highlight several important features.

[1] A prefix for a decryption query is defined as the common prefix blocks between the decryption query input string and an encryption query output string. The length of the common prefix for the i^{th} decryption query is denoted as p_i. Note that, if the decryption query uses a fresh nonce, then the decryption query input string does not share any common prefix with any of the encryption query output strings and we set $p_i = -1$.

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} [33], 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-128 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 [31] 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 round key 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.

4.4 New third-party analysis and its implications

Besides the security argument by the designers, GIFT-128 has received a lot of third-party analysis. Moreover, during the first and second rounds, several groups analyzed the security of GIFT-COFB. Here, we summarize the third-party analysis against GIFT-128 and GIFT-COFB, which suggests that GIFT-COFB is highly secure against cryptanalysis.

4.4.1 Third-party analysis on GIFT-128

In short, our underlying 40-round block cipher GIFT-128 [5] remains secure with high security margin. We have summarized the latest third-party cryptanalysis results in Table 4.2.

[33] is the corrected version of [34] with the 22-round differential cryptanalysis on GIFT-128, the original 23-round attack was invalid.

Although GIFT-128 did not make related-key security claims, third-party analysis [10, 18, 23] have shown that GIFT-128 is actually resistant against related-key attacks.

4.4.2 Third-party analysis on GIFT-COFB

Zong et al. [35] applied their linear cryptanalysis to mount the key-recovery attack on the reduced-round variant of GIFT-COFB, in which the number of rounds of GIFT-128 is reduced to 15 rounds. In short, it makes many encryption queries under different nonces to obtain pairs of plaintext and ciphertext in the consequent two blocks. The pairs partially reveal the internal state value. By setting the linear masks only to exploit those values, linear cryptanalysis can be mounted. The attack complexity is $(Time, Data, Memory) = (2^{90.7}, 2^{62}, 2^{96})$. Note that the number of attacked rounds is significantly smaller than that of GIFT-128, because of the limited degrees of freedom for the attacker to set the active bit positions. Also note that Zong et al. [35] show that the similar attack can be mounted on SUNDAE-GIFT up to 16 rounds, 1 round longer than GIFT-COFB because of the difference of the bit-positions to extract the key stream. This illustrates the validity of GIFT-COFB on the bit-positions of extracting the key stream.

Khairallah analyzed the security of GIFT-COFB as a mode [19, 20], i.e., GIFT-128 is treated as a black box. In [19], a forgery attack against GIFT-COFB that makes $O(2^{n/2})$ encryption queries and $O(2^{n/2})$ decryption queries in a single key setting is presented. An analysis in the multi-key setting is also presented. In [19], the forgery attack is improved to make $O(2^{n/4})$ encryption queries and $O(2^{n/2})$ decryption queries. These attacks are almost matching attacks to the provable security bound, up to the logarithmic factor. That is, these results show that the provable security bound presented as Eq. (4.1) is almost tight.

There was a paper posted on Cryptology ePrint Archive 2020/698 [11] claiming forgery attack on GIFT-COFB, but we have contacted and clarified with the authors that the attack is invalid due to an oversight of the GIFT-COFB specification and the authors have since been withdrawn their paper.

4.4.3 Third-party analysis from various viewpoints

In addition to conventional cryptanalysis, GIFT-128 receives third-party evaluation from different viewpoints.

Hou et al. [14] investigated physical security of GIFT-COFB, in particular differential ciphertext side-channel attacks.

Jang et al. [15] and Bijwe et al. [7] evaluated the post-quantum security of GIFT-128, in particular, amount of quantum resource to implement the Grover search on GIFT-128.

Table 4.2: Summary of third-party analysis result on `GIFT-128`. Rounds with asterisk (*) are optimal results. SK – single-key, RK – related-key, LC – linear cryptanalysis, DC – differential cryptanalysis.

Setting	Rounds	Approach	Prob.	Time	Data	Mem.	Ref.
Distinguisher							
SK	11	Integral	1	-	2^{127}	-	[13]
SK	9*	LC	2^{-44}	-	-	-	[17]
SK	10*	LC	2^{-52}	-	-	-	[17]
SK	15	LC	2^{-109}	-	-	-	[35]
SK	9*	DC	$2^{-45.4}$	-	-	-	[22]
SK	10*	DC	$2^{-49.4}$	-	-	-	[22]
SK	11*	DC	$2^{-54.4}$	-	-	-	[22]
SK	12*	DC	$2^{-60.4}$	-	-	-	[22]
SK	13*	DC	$2^{-67.8}$	-	-	-	[22]
SK	14*	DC	$2^{-79.000}$	-	-	-	[17]
SK	15*	DC	$2^{-85.415}$	-	-	-	[17]
SK	16*	DC	$2^{-90.415}$	-	-	-	[17]
SK	17*	DC	$2^{-96.415}$	-	-	-	[17]
SK	18	DC	2^{-109}	-	-	-	[33]
SK	18*	DC	$2^{-103.415}$	-	-	-	[17]
SK	19	DC	$2^{-110.83}$	-	-	-	[17]
SK	20	DC	$2^{-121.415}$	-	-	-	[21]
SK	20	DC	$2^{-120.245}$	-	-	-	[18]
SK	20	DC	$2^{-121.813}$	-	-	-	[35]
SK	21	DC	$2^{-126.4}$	-	-	-	[22]
RK	7	DC	$2^{-15.83}$	-	-	-	[10]
RK	10	DC	$2^{-72.66}$	-	-	-	[10]
RK	19	Boomerang	$2^{-121.2}$	-	-	-	[23]
RK	19	Boomerang	$2^{-109.626}$	-	-	-	[18]
Key-Recovery							
SK	22	LC	-	2^{117}	2^{117}	2^{78}	[35]
SK	22	DC	-	2^{114}	2^{114}	2^{53}	[33]
SK	26	DC	-	$2^{124.415}$	2^{109}	2^{109}	[21]
SK	26	DC	-	$2^{123.245}$	$2^{123.245}$	2^{109}	[18]
SK	27	DC	-	$2^{124.83}$	$2^{123.53}$	2^{80}	[35]
RK	21	Boomerang	-	$2^{126.6}$	$2^{126.6}$	$2^{126.6}$	[23]
RK	22	Boomerang	-	$2^{112.63}$	$2^{112.63}$	2^{52}	[18]
RK	23	Rectangle	-	$2^{126.89}$	$2^{121.31}$	$2^{121.63}$	[18]

Chapter 5

Design Rationale

5.1 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 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 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.

5.2 Block Cipher Primitive: GIFT-128

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 [8]. 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-128` when we consider software implementation. Here, we will state the 3 most obvious benefits relating to its 3 steps in a round function.

Firstly for the non-linear layer, for LUT based implementation, we can consider updating 2 `GIFT-128` 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.

Secondly for the linear layer, while it is basically free on hardware, for software implementation it was extremely slow and complex to implement. However in 2020, a technique called fix-slicing [2] was introduced for bitslice implementation and it significantly improves the constant-time implementation of `GIFT-128` on 32-bit architectures. As a result, bitslice implementation is much more efficient that LUT based implementation.

Third and lastly the key addition, for LUT based implementation, the round keys 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 round keys, 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.

5.3 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 bit.
- 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, this updates does not have impact on the security of the mode (only a nominal 1-bit security degradation).

5.4 Proposal for hash functionality

If hash functionality is desired, we propose constructing a 256-bit hash function using `GIFT-128` in the double block length (DBL) hashing construction proposed by Mennink [25] ("Mennink's construction" for short) for two reasons.

Firstly, it is a well-established construction. Mennink's construction was first proposed at ASIACRYPT 2012 [24], and to the best of our knowledge, after almost a decade, it remains secure.

Secondly, it is suitable for our primitive block cipher `GIFT-128` and provides sufficiently high security guarantee. Mennink's construction uses block ciphers with n-bit block size and n-bit key (also known as DBL^n functions), since `GIFT-128` is a 128-bit block cipher with 128-bit key, it is a natural candidate. In addition, Mennink's construction provides the highest collision (2^n) and preimage ($2^{3n/2}$) security among other the known DBL^n functions.

Acknowledgments

Subhadeep Banik is supported by the Ambizione grant PZ00P2_179921, awarded by the Swiss National Science Foundation. Thomas Peyrin is supported by the Temasek Labs grant (DSOCL16194).

Bibliography

[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, Andrey Bogdanov, Francesco Regazzoni, Takanori Isobe, Harunaga Hiwatari, and Toru Akishita. Inverse gating for low energy encryption. In *HOST*, pages 173–176. IEEE Computer Society, 2018.

[4] Subhadeep Banik, Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, Mridul Nandi, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT-COFB. *IACR Cryptol. ePrint Arch.*, 2020:738, 2020.

[5] 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.

[6] 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`.

[7] Subodh Bijwe, Amit Kumar Chauhan, and Somitra Kumar Sanadhya. Quantum Search for Lightweight Block Ciphers: GIFT, SKINNY, SATURNIN. *IACR Cryptol. ePrint Arch.*, 2020:1485, 2020.

[8] 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.

[9] Andrea Caforio, Fatih Balli, and Subhadeep Banik. Energy analysis of lightweight AEAD circuits. Accepted in Ceryptography and Network Security (CANS) 2020.

[10] Meichun Cao and Wenying Zhang. Related-Key Differential Cryptanalysis of the Reduced-Round Block Cipher GIFT. *IEEE Access*, 7:175769–175778, 2019.

[11] Zhe CEN, Xiutao FENG, Zhangyi Wang, and Chunping CAO. (–Withdrawn–) Forgery attack on the authentication encryption GIFT-COFB. Cryptology ePrint Archive, Report 2020/698, 2020. `https://eprint.iacr.org/2020/698`.

[12] 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.

[13] Zahra Eskandari, Andreas Brasen Kidmose, Stefan Kölbl, and Tyge Tiessen. Finding Integral Distinguishers with Ease. In *SAC*, volume 11349 of *Lecture Notes in Computer Science*, pages 115–138. Springer, 2018.

[14] Xiaolu Hou, Jakub Breier, and Shivam Bhasin. DNFA: Differential No-Fault Analysis of Bit Permutation Based Ciphers Assisted by Side-Channel. In *DATE*, 2021. to appear. The preprint version is available at IACR Cryptol. ePrint Arch. 2020/1554.

[15] Kyoungbae Jang, Hyunjun Kim, Siwoo Eum, and Hwajeong Seo. Grover on GIFT. *IACR Cryptol. ePrint Arch.*, 2020:1405, 2020.

[16] Arpan Jati, Naina Gupta, Anupam Chattopadhyay, Somitra Kumar Sanadhya, and Donghoon Chang. Threshold implementations of GIFT: A trade-off analysis. *IEEE Trans. Inf. Forensics Secur.*, 15:2110–2120, 2020.

[17] Fulei Ji, Wentao Zhang, and Tianyou Ding. Improving Matsui's Search Algorithm for the Best Differential/Linear Trails and its Applications for DES, DESL and GIFT. *The Computer Journal*, 64(4):610–627, April 2021. available at IACR Cryptol. ePrint Arch. 2019/1190.

[18] Fulei Ji, Wentao Zhang, Chunning Zhou, and Tianyou Ding. Improved (Related-key) Differential Cryptanalysis on GIFT. In *SAC*, Lecture Notes in Computer Science. Springer, 2021. to appear. The preprint version is available at IACR Cryptol. ePrint Arch. 2020/1242.

[19] Mustafa Khairallah. Weak Keys in the Rekeying Paradigm: Application to COMET and mixFeed. *IACR Trans. Symmetric Cryptol.*, 2019(4):272–289, 2019.

[20] Mustafa Khairallah. Observations on the Tightness of the Security Bounds of GIFT-COFB and HyENA. *IACR Cryptol. ePrint Arch.*, 2020:1463, 2020.

[21] Lingchen Li, Wenling Wu, Yafei Zheng, and Lei Zhang. The Relationship between the Construction and Solution of the MILP Models and Applications. *IACR Cryptol. ePrint Arch.*, 2019:49, 2019.

[22] Yu Liu, Huicong Liang, Muzhou Li, Luning Huang, Kai Hu, Chenhe Yang, and Meiqin Wang. STP Models of Optimal Differential and Linear Trail for S-box Based Ciphers. *Science China Information Sciences*, 64(159103), May 2021. available at IACR Cryptol. ePrint Arch. 2019/25.

[23] Yunwen Liu and Yu Sasaki. Related-Key Boomerang Attacks on GIFT with Automated Trail Search Including BCT Effect. In *ACISP*, volume 11547 of *Lecture Notes in Computer Science*, pages 555–572. Springer, 2019.

[24] Bart Mennink. Optimal collision security in double block length hashing with single length key. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 526–543. Springer, 2012.

[25] Bart Mennink. Optimal collision security in double block length hashing with single length key. *Des. Codes Cryptogr.*, 83(2):357–406, 2017.

[26] NIST. Lightweight cryptography project, 2019.

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

[28] 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.

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

[30] 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.

[31] 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.

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

[33] 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`.

[34] Baoyu Zhu, Xiaoyang Dong, and Hongbo Yu. Milp-based differential attack on round-reduced GIFT. In *CT-RSA*, volume 11405 of *Lecture Notes in Computer Science*, pages 372–390. Springer, 2019.

[35] Rui Zong, Xiaoyang Dong, Huaifeng Chen, Yiyuan Luo, Si Wang, and Zheng Li. Towards Key-recovery-attack Friendly Distinguishers: Application to GIFT-128. *IACR Trans. Symmetric Cryptol.*, 2021(1):156–184, 2021.

Changelog

- 29-03-2019: version v1.0
- 17-05-2021: version v1.1. Updated Chapter 3 with latest performance results. Revised Chapter 4 for clarifications and added third-party analysis on GIFT-COFB. Added a hash function proposal in Chapter 5. (No change in GIFT-COFB specification.)