# Thank Goodness It's Friday (TGIF)

v1.0

**Designers/Submitters (in alphabetical order):**


Tetsu Iwata[1], Mustafa Khairallah[2], Kazuhiko Minematsu[3]
Thomas Peyrin[2], Yu Sasaki[4], Siang Meng Sim[2], Ling Sun[5]

[1] Nagoya University, Japan
tetsu.iwata@nagoya-u.jp

[2] Nanyang Technological University, Singapore
mustafam001@e.ntu.edu.sg,
thomas.peyrin@ntu.edu.sg,
crypto.s.m.sim@gmail.com

[3] NEC Corporation, Japan
k-minematsu@ah.jp.nec.com

[4] NTT, Japan
sasaki.yu@lab.ntt.co.jp

[5] Shandong University, China
lingsun@mail.sdu.edu.cn

# Contents

# Introduction

This document specifies TGIF, an authenticated encryption with associated data (AEAD) scheme based on a tweakable block cipher (TBC) TGIF-TBC. TGIF consists of two families, a nonce-based AE (NAE) TGIF-N and a nonce misuse-resistant AE (MRAE) TGIF-M. TGIF aims at lightweight, efficient, and highly-secure NAE and MRAE schemes, based on a TBC.

As the underlying TBC, we propose TGIF-TBC, a new lightwieght TBC strongly inspired from GIFT cipher. More precisely, TGIF-TBC can be seen as an improvement over the 128-bit version of GIFT [2] cipher, by providing improved performances, improved security guarantees, and enabling tweak capabilities. We end up with an extremely efficient cipher, from very constrained hardware to high-end software.

The operating mode used in TGIF is taken from the `Remus` [12] submission, which strongly minimizes the area overhead on top of the internal TBC. It provides full $n$-bit security and can very easily switch from nonce-respecting mode to nonce-misuse resistant mode.

**Organization of the document.** In Section 2, we first introduce the basic notations and the notion of tweakable block cipher, followed by the list of parameters for TGIF, the recommended parameter sets, and the specification of TBC TGIF-TBC. In the last part of Section 2, we specify two families of TGIF, TGIF-N and TGIF-M. We present our security claims in Section 3 and show our security analysis of TGIF-TBC in Section 4. In Section 5, we describe the desirable features of TGIF. The design rationale under our schemes, including some details of modes and choice of the TBC, is presented in Section 6. Finally, we show some implementation aspects of TGIF in Section 7.

# Specification

## 2.1 Notations

Let $\{0,1\}^*$ be the set of all finite bit strings, including the empty string $\varepsilon$. For $X \in \{0,1\}^*$, let $|X|$ denote its bit length. Here $|\varepsilon| = 0$. For integer $n \geq 0$, let $\{0,1\}^n$ be the set of $n$-bit strings, and let $\{0,1\}^{\leq n} = \bigcup_{i=0,\ldots,n} \{0,1\}^i$, where $\{0,1\}^0 = \{\varepsilon\}$. Let $[\![n]\!] = \{1,\ldots,n\}$ and $[\![n]\!]_0 = \{0,1,\ldots,n-1\}$.

For two bit strings $X$ and $Y$, $X \,\|\, Y$ is their concatenation. We also write this as $XY$ if it is clear from the context. Let $0^i$ be the string of $i$ zero bits, and for instance we write $10^i$ for $1 \,\|\, 0^i$. We denote $\mathtt{msb}_x(X)$ (resp. $\mathtt{lsb}_x(X)$) the truncation of $X$ to its $x$ most (resp. least) significant bits. See"Endian" paragraph below. Bitwise XOR of two variables $X$ and $Y$ is denoted by $X \oplus Y$, where $|X| = |Y| = c$ for some integer $c$. By convention, if one of $X$ or $Y$ is represented as an integer in $[\![2^c]\!]_0$ we assume a standard integer-to-binary encoding: for example $X \oplus 1$ denotes $X \oplus 0^{c-1}1$.

**Padding.** For $X \in \{0,1\}^{\leq l}$ of length multiple of 8 (i.e. byte string),

$$\mathtt{pad}_l(X) = \begin{cases} X & \text{if } |X| = l, \\ X \,\|\, 0^{l-|X|-8} \,\|\, \mathtt{len}_8(X), & \text{if } 0 \leq |X| < l, \end{cases}$$

where $\mathtt{len}_8(X)$ denotes the one-byte encoding of the byte-length of $X$. Here, $\mathtt{pad}_l(\varepsilon) = 0^l$. When $l = 128$, $\mathtt{len}_8(X)$ has 16 variations (*i.e.*, byte length 0 to 15), and we encode it to the last 4 bits of $\mathtt{len}_8(X)$ (for example, $\mathtt{len}_8(11) = \mathtt{00001011}$). The case $l = 64$ is similarly treated, by using the last 3 bits.

**Parsing.** For $X \in \{0,1\}^*$, let $|X|_n = \max\{1, \lceil |X|/n \rceil\}$. Let $(X[1],\ldots,X[x]) \xleftarrow{n} X$ be the parsing of $X$ into $n$-bit blocks. Here $X[1] \,\|\, X[2] \,\|\, \ldots \,\|\, X[x] = X$ and $x = |X|_n$. When $X = \varepsilon$ we have $X[1] \xleftarrow{n} X$ and $X[1] = \varepsilon$. Note in particular that $|\varepsilon|_n = 1$.

**Galois Field.** An element $a$ in the Galois field $\mathrm{GF}(2^n)$ will be interchangeably represented as an $n$-bit string $a_{n-1}\ldots a_1 a_0$, a formal polynomial $a_{n-1}\mathtt{x}^{n-1} + \cdots + a_1\mathtt{x} + a_0$, or an integer $\sum_{i=0}^{n-1} a_i 2^i$.

**Matrix.** Let $G$ be an $n \times n$ binary matrix defined over $\mathrm{GF}(2)$. For $X \in \{0,1\}^n$, let $G(X)$ denote the matrix-vector multiplication over $\mathrm{GF}(2)$, where $X$ is interpreted as a column vector. We may write $G \cdot X$ instead of $G(X)$.

**Endian.** We employ little endian for byte ordering: a $b$-bit string $X$ is received as

$$X_7 X_6 \ldots X_0 \,\|\, X_{15} X_{14} \ldots X_8 \,\|\, \ldots \,\|\, X_{n-1} X_{n-2} \ldots X_{n-8},$$

where $X_i$ denotes the $(i+1)$-st bit of $X$ (for $i \in [\![b]\!]_0$). Therefore, when $c$ is a multiple of 8 and $X$ is a byte string, $\mathtt{msb}_c(X)$ and $\mathtt{lsb}_c(X)$ denote the last (rightmost) $c$ bytes of $X$ and the first

(leftmost) $c$ bytes of $X$, respectively. For example, $\texttt{lsb}_{16}(X) = (X_7X_6 \dots X_0 \,\|\, X_{15}X_{14} \dots X_8)$ and $\texttt{msb}_8(X) = (X_{n-1}X_{n-2} \dots X_{n-8})$ with the above $X$. Since our specification is defined over byte strings, we only consider the above case for `msb` and `lsb` functions (*i.e.*, the subscript $c$ is always a multiple of 8).

**(Tweakable) Block Cipher.** A tweakable block cipher (TBC) is a keyed function $\widetilde{E} : \mathcal{K} \times \mathcal{T}_{\mathcal{W}} \times \mathcal{M} \to \mathcal{M}$, where $\mathcal{K}$ is the key space, $\mathcal{T}_{\mathcal{W}}$ is the tweak space, and $\mathcal{M} = \{0,1\}^n$ is the message space, such that for any $(K, T_w) \in \mathcal{K} \times \mathcal{T}_{\mathcal{W}}$, $\widetilde{E}(K, T_w, \cdot)$ is a permutation over $\mathcal{M}$. We interchangeably write $\widetilde{E}(K, T_w, M)$ or $\widetilde{E}_K(T_w, M)$ or $\widetilde{E}_K^{T_w}(M)$. When $\mathcal{T}_{\mathcal{W}}$ is singleton, it is essentially a block cipher and is simply written as $E : \mathcal{K} \times \mathcal{M} \to \mathcal{M}$.

## 2.2 Parameters

TGIF has the following parameters:

- Nonce length $nl = 128$.
- Key length $k = 128$.
- Message and AD block length $n = 128$.
- Mode to convert a block cipher into a TBC, $\texttt{ICmode} \in \{\mathsf{ICE1}, \mathsf{ICE2}\}$.
- Counter bit length $d = 128$. *Counter* refers the part of the tweakey that changes after each TBC call, for the same $(N, K)$ pair. Each variants of TGIF has $2^d - 1$ possible counter values for each $(N, K)$ pair.
- Tag length $\tau = n$.

The block cipher $E : \mathcal{K} \times \mathcal{M} \to \mathcal{M}$ with $\mathcal{M} = \{0,1\}^{128}$ and $\mathcal{K} = \{0,1\}^{128}$. Here, $E$ is TGIF-TBC, by seeing the whole tweakey space as the key space, and assuming certain tweakey encodings specified in Section 2.4.

While our submission fixes $\tau = n$, a tag for NAE schemes can be truncated if needed (not for MRAE), at the cost of decreased security against forgery. See Section 4.

**NAE and MRAE families.** TGIF has two families, TGIF-N and TGIF-M, and each family consists of two members (the sets of parameters). The former implements nonce-based AE (NAE) secure against Nonce-respecting adversaries, and the latter implements nonce Misuse-resistant AE (MRAE) introduced by Rogaway and Shrimpton [26]. The name TGIF stands for the set of two families.

## 2.3 Recommended Parameter Sets

We present our members (parameters) in Table 2.1. The primary member of our submission is TGIF-N1. All members conform to the requirements of NIST call for proposal with respect to key length, nonce length, and maximum input length.

## 2.4 The Tweakable Block Cipher TGIF-TBC

In this section, we describe our tweakable block-cipher TGIF-TBC. It has a single version: the tweakey state is 128 bits, while the plaintext/ciphertext is also 128 bits. We omit the description of its inverse as all its component are trivial to invert.

TGIF-TBC is an iterative Substitution-Permutation Network (SPN) based TBC, composed of 18 steps. It is largely inspired by GIFT-64 and GIFT-128 ciphers [2]. Indeed, TGIF-TBC will reuse 4

**Table 2.1:** Members of TGIF.

| Family | Name | $E$ | ICmode | $k$ | $nl$ | $n$ | $d$ | $\tau$ |
|---|---|---|---|---|---|---|---|---|
| TGIF-N | TGIF-N1 | TGIF-TBC | ICE1 | 128 | 128 | 128 | 128 | 128 |
| | TGIF-N2 | TGIF-TBC | ICE2 | 128 | 128 | 128 | 128 | 128 |
| TGIF-M | TGIF-M1 | TGIF-TBC | ICE1 | 128 | 128 | 128 | 128 | 128 |
| | TGIF-M2 | TGIF-TBC | ICE2 | 128 | 128 | 128 | 128 | 128 |

rounds of GIFTb as a black box, while GIFTb is a slight variation of GIFT-64 where the data arrives in bitslice form (the (twea)key schedule is also changed). There are different ways to perceive GIFT-64, but the more pictorial description is detailed in the GIFT paper, which looks like a larger version of PRESENT cipher with 32 4-bit S-boxes and an 128-bit bit permutation (see Figure **??**).

### Initialisation

The 128-bit state of TGIF-TBC is thus viewed as two words $L$ and $R$ of 64-bit each, each of these words seen in turn as four 16-bit words $L_0$, $L_1$, $L_2$, $L_3$, $R_0$, $R_1$, $R_2$ and $R_3$. We load the incoming plaintext bytes $B_0, \ldots, B_{15}$ in $L_0$ first, then $L_1$, etc., finishing with $R_3$.

$$L = \begin{cases} L_0 & = B_1 || B_0 \\ L_1 & = B_3 || B_2 \\ L_2 & = B_5 || B_4 \\ L_3 & = B_7 || B_6 \end{cases} \qquad R = \begin{cases} R_0 & = B_9 \,||\, B_8 \\ R_1 & = B_{11} || B_{10} \\ R_2 & = B_{13} || B_{12} \\ R_3 & = B_{15} || B_{14} \end{cases}$$

The 128-bit tweakey state $KS$ is viewed as four words $TK_0$, $TK_1$, $TK_2$ and $TK_3$ of 32-bit each. We load the incoming tweakey bytes $K_0, \ldots, K_{15}$ in $TK_0$ first, then $TK_1$, etc., finishing with $TK_3$.

$$TK = \begin{cases} TK_0 & = K_3 \,||K_2\, ||K_1\, ||K_0 \\ TK_1 & = K_7 \,||K_6\, ||K_5\, ||K_4 \\ TK_2 & = K_{11} || K_{10} || K_9 \,||B_8 \\ TK_3 & = K_{15} || K_{14} || K_{13} || K_{12} \end{cases}$$

### Round function of GIFTb

TGIF-TBC will use 4 rounds of a bitslice version of GIFT-64 as black box (that is, GIFT-64 with the data arriving in bitslice mode instead, we denote this cipher GIFTb), denoted $\mathsf{GIFTb}_4(S, TK)$ where $S$ is a 64-bit state composed of four 16-bit words $S_0$, $S_1$, $S_2$ and $S_3$, and where $TK$ is the 128-bit tweakey state. Each round of GIFT-64 consists of 4 steps: AddRoundTweakey, AddConstant, SubCells and PermBits. We use a new bitslice representation to

AddRoundTweakey. This step consists of adding the subtweakey word $TK_0$ to $S_0$ and to $S_1$: $S_0 = S_0 \oplus TK_0$ and $S_1 = S_1 \oplus (TK_0) >> 16$. Then the tweakey schedule function is applied to the tweakey state (see description below).

AddConstant. This step consists of XORing the least significant bits of state $S_3$ with 7-bit round dependant constants $c_i$ (given in Table **??**) and swapping the most significant bit of $S_3$ : $S_3 = S_3 \oplus c_i \oplus \texttt{0x8000}$. These constants have been generated with a 7-bit maximum-length LFSR initialised to 1. Let $x_6, x_5, x_4, x_3, x_2, x_1, x_0$ be the input seven bits ($x_0$ being the LSB), then after clocking the LFSR we obtain: $x_5, x_4, x_3, x_2, x_1, x_0 \oplus x_6, x_6$.

| Step | Constants (round 0, round 1, round 2, round 3) | | | |
|---|---|---|---|---|
| 0 - 3 | `01,02,04,08` | `10,20,40,03` | `06,0c,18,30` | `60,43,05,0a` |
| 4 - 7 | `14,28,50,23` | `46,0f,1e,3c` | `78,73,65,49` | `11,22,44,0b` |
| 8 - 11 | `16,2c,58,33` | `66,4f,1d,3a` | `74,6b,55,29` | `52,27,4e,1f` |
| 12 - 15 | `3e,7c,7b,75` | `69,51,21,42` | `07,0e,1c,38` | `70,63,45,09` |
| 16 - 17 | | `12,24,48,13` | `26,4c,1b,36` | |

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 16-bit wise `AND`, `OR` and `NOT` operations respectively.

PermBits. Different 16-bit bit permutations are applied to each $S_i$ independently.

In the first round, we apply to $S_i$ a right rotation of $i$ positions inside each 4-bit subwords (note that $S_0$ is thus not modified):

$$\begin{aligned} S_0 &: (15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0) \mapsto (15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0) \\ S_1 &: (15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0) \mapsto (12,15,14,13,8,11,10,9,4,7,6,5,0,3,2,1) \\ S_2 &: (15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0) \mapsto (13,12,15,14,9,8,11,10,5,4,7,6,1,0,3,2) \\ S_3 &: (15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0) \mapsto (14,13,12,15,10,9,8,11,6,5,4,7,2,1,0,3) \end{aligned}$$

In the second round, we apply to $S_i$ a right rotation of $4i$ positions inside the entire 16-bit word (note that $S_0$ is thus not modified):

$$\begin{aligned} S_0 &: (15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0) \mapsto (15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0) \\ S_1 &: (15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0) \mapsto (3,2,1,0,15,14,13,12,11,10,9,8,7,6,5,4) \\ S_2 &: (15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0) \mapsto (7,6,5,4,3,2,1,0,15,14,13,12,11,10,9,8) \\ S_3 &: (15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0) \mapsto (11,10,9,8,7,6,5,4,3,2,1,0,15,14,13,12) \end{aligned}$$

In the third round, we apply to $S_i$ a left rotation of $i$ positions inside each 4-bit subwords (note that $S_0$ is thus not modified):

$$\begin{aligned} S_0 &: (15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0) \mapsto (15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0) \\ S_1 &: (15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0) \mapsto (14,13,12,15,10,9,8,11,6,5,4,7,2,1,0,3) \\ S_2 &: (15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0) \mapsto (13,12,15,14,9,8,11,10,5,4,7,6,1,0,3,2) \\ S_3 &: (15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0) \mapsto (12,15,14,13,8,11,10,9,4,7,6,5,0,3,2,1) \end{aligned}$$

In the fourth round, we apply to $S_i$ a left rotation of $4i$ positions inside the entire 16-bit word (note that $S_0$ is thus not modified):

$$S_0 : (15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0) \mapsto (15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0)$$
$$S_1 : (15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0) \mapsto (11,10,9,8,7,6,5,4,3,2,1,0,15,14,13,12)$$
$$S_2 : (15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0) \mapsto (7,6,5,4,3,2,1,0,15,14,13,12,11,10,9,8)$$
$$S_3 : (15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0) \mapsto (3,2,1,0,15,14,13,12,11,10,9,8,7,6,5,4)$$

There is an interesting property that after these 4 rounds of permutations, the bits in each $S_i$ returns to its initial position. E.g. For $S_1$, applying the 4 rounds of operation, we get

$$\begin{aligned} S_1 : (15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0) &\mapsto (12,15,14,13,8,11,10,9,4,7,6,5,0,3,2,1) \\ &\mapsto (0,3,2,1,12,15,14,13,8,11,10,9,4,7,6,5) \\ &\mapsto (3,2,1,0,15,14,13,12,11,10,9,8,7,6,5,4) \\ &\mapsto (15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0) \end{aligned}$$

### Step function of TGIF-TBC

The step function applies iteratively a Misty structure [**?**]. Let $L$ and $R$ be the two 64-bit left and right internal states, then each step of TGIF-TBC will perform:

$$\begin{cases} L' &= \mathsf{GIFTb}_4(R, TK) \oplus (L \lll 55) \\ R' &= L \end{cases}$$

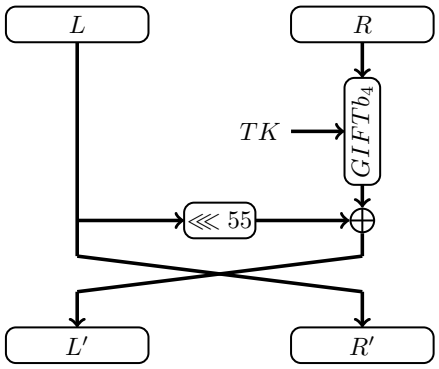


**Figure 2.1:** The step function of TGIF-TBC.

where $TK$ is the current tweakey state. We recall that the tweakey state is updated every round inside $\mathsf{GIFTb}_4$ with the tweakey schedule.

### Tweakey schedule of TGIF-TBC

The tweakey state, decomposed as $TK = TK_0||TK_1||TK_2||TK_3$ is updated as follows:

$$\begin{cases} TK'_0 &= TK_1 \\ TK'_1 &= TK_2 \\ TK'_2 &= TK_3 \\ TK'_3 &= TK_0 \oplus TK_1 \oplus ((TK_3 \oplus 1) \lll 8) \oplus 1 \end{cases}$$

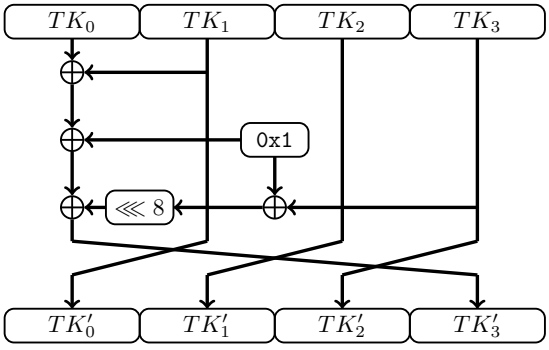


**Figure 2.2:** The TGIF-TBC tweakey schedule

This is basically a simple generalised Feistel construction with a linear update function. It has the special property to cycle every 24 calls, and will thus cycle at the end of the cipher (every 72 rounds, or 18 steps). In other words, the first and last key states are identical.

**Finalisation**

The final value of the internal state array provides the ciphertext with cells being unpacked in the same way as the packing during initialization. Test vectors for TGIF-TBC are provided below.

```
/* TGIF-TBC */
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:  F6 1B 94 03 28 C7 58 B6 90 A6 5A B1 03 4B 9B B7

Key:         3A E8 AF 4B C5 88 01 5C 24 C6 48 BC 2C 1C CD E5
Plaintext:   AC 3B C6 DC 0D 10 08 1E CE 8C B4 E3 28 B2 DD D5
Ciphertext:  2E A6 0D BE F6 38 09 F9 80 34 F4 AF 3A DE 28 D9
```

## 2.5 The Authenticated Encryption TGIF

### 2.5.1 Block Counters and Domain Separation

**Domain separation.** We will use a domain separation byte $B$ to ensure appropriate independence between the tweakable block cipher calls and the various versions of TGIF. Let $B = (b_7\|b_6\|b_5\|b_4\|b_3\|b_2\|b_1\|b_0)$ be the bitwise representation of this byte, where $b_7$ is the MSB and $b_0$ is the LSB (see also Figure 2.3). Then, we have the following:

- $b_6b_5$ will specify the parameter sets. They are fixed to:
  - 00 for TGIF-N1
  - 01 for TGIF-M1
  - 10 for TGIF-N2
  - 11 for TGIF-M2

  Note that all nonce-respecting modes have $b_5 = 0$ and all nonce-misuse resistant modes have $b_5 = 1$.
- $b_7$ and $b_4$ are set to 0.
- $b_3$ is set to 1 once we have handled the last block of data (AD and message chains are treated separately), to 0 otherwise.
- $b_2$ is set to 1 when we are performing the authentication phase of the operating mode (i.e. when no ciphertext data is produced), to 0 otherwise. In the special case where $b_5 = 1$ and $b_4 = 1$ (i.e. last block for the nonce-misuse mode), $b_3$ will instead denote if the number of message blocks is even ($b_5 = 1$ if that is the case, 0 otherwise).
- $b_1$ is set to 1 when we are handling a message block, to 0 otherwise. Note that in the case of the misuse-resistant modes, the message blocks will be used during authentication phase (in which case we will have $b_3 = 1$ and $b_2 = 1$). In the special case where $b_5 = 1$ and $b_4 = 1$ (i.e. last block for the nonce-misuse mode), $b_3$ will instead denote if the number of message blocks is even ($b_5 = 1$ if that is the case, 0 otherwise).
- $b_0$ is set to 1 when we are handling a padded block (associated data or message), to 0 otherwise.

The reader can refer to Table 7.1 in the Appendix to obtain the exact specifications of the domain separation values depending on the various cases.

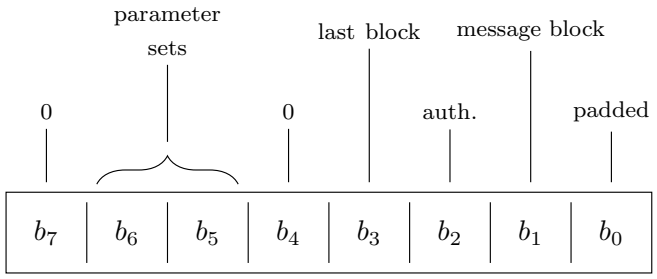


**Figure 2.3:** Domain separation when using the tweakable block cipher

**Doubling over a Finite Field.** For any positive integer $c$, we assume $\mathrm{GF}(2^c)$ is defined over the lexicographically-first polynomial among the irreducible degree $c$ polynomials of a minimum number of coefficients. We use single field: $\mathrm{GF}(2^c)$ for $c = 128$. The primitive polynomial is:

$$\mathtt{x}^{128} + \mathtt{x}^7 + \mathtt{x}^2 + \mathtt{x} + 1 \text{ for } c = 128. \tag{2.1}$$

Let $Z = (z_{c-1} z_{c-2} \dots z_1 z_0)$ for $z_i \in \{0,1\}$, $i \in [\![c]\!]_0$ be an element of $\mathrm{GF}(2^c)$. A multiplication of $Z$ by the generator (polynomial $\mathtt{x}$) is called *doubling* and written as $2Z$ [24]. An $i$-times doubling of $Z$ is written as $2^i Z$, and is efficiently computed from $2^{i-1}Z$ (see below). Here, $2^0 Z = Z$ for any $Z$. When $Z = 0^n$, *i.e.*, zero entity in the field, then $2^i Z = 0^n$ for any $i \geq 0$.

Alternatively, let $X \ll i$ as the $i$-bit left shift of $X$, *i.e.*, $x_i \leftarrow x_{i-1}\ \forall\ i \in [\![|X| - 1]\!] \setminus \{0\}, x_0 \leftarrow 0$. Then $2Z = (Z \ll 1)$ if $z_{c-1} = 0$, otherwise, $2Z = (Z \ll 1) \oplus \mathtt{const}_c$, where $\mathtt{const}_{128} = (10000111 \,\|\, 0^{120})$ in little endian.

To avoid confusion, we may write $\overline{D}$ (in particular when it appears in a part of tweak) in order to emphasize that this is indeed a doubling-based counter, *i.e.*, $2^D X$ for some key-dependent variable $X$. One can interpret $\overline{D}$ as $2^D$ (but in that case it is a part of tweakey state or a coefficient of mask, and *not* a part of input of ICE).

On bit-level, doubling $Z \to 2Z$ over $\mathrm{GF}(2^c)$ for $c = 128$ is defined as

$$\begin{aligned}
&z_i \leftarrow z_{i-1} \text{ for } i \in [\![127]\!] \setminus \{7, 2, 1, 0\},\\
&z_7 \leftarrow z_6 \oplus z_{127},\\
&z_2 \leftarrow z_1 \oplus z_{127},\\
&z_1 \leftarrow z_0 \oplus z_{127},\\
&z_0 \leftarrow z_{127}.
\end{aligned}$$

### 2.5.2 The TBC ICE

In the specification of TGIF, TGIF-TBC is not directly used as a TBC. Instead, we use TGIF-TBC as a building block for the TBC which we call ICE (for Ideal-Cipher Encryption), defined in Figure 2.4. $\mathsf{ICE} : (\mathcal{L} \times \mathcal{V}) \times (\mathcal{D} \times \mathcal{B}) \times \mathcal{M} \to \mathcal{M}$ is defined with $\mathcal{L} = \{0,1\}^l$, where $l = 128$, and $\mathcal{V} = \mathcal{M} = \{0,1\}^n$, and it has 2 variants, ICE1 and ICE2. The counter space $\mathcal{D}$ is $[\![2^d - 1]\!]$ and $d$ follows from Table. 2.1 and domain byte $\mathcal{B}$ is as defined earlier. In each ICE variant, $\mathsf{TGIF\text{-}TBC} : \mathcal{K} \times \mathcal{M} \to \mathcal{M}$ is used as the internal TBC (in fact as a block cipher), and there is an encode function $\mathtt{encode} : \mathcal{L} \times \mathcal{D} \times \mathcal{B} \to \mathcal{K}$ used to derive a $k$-bit tweakey (for $k = 128$) $T_K$ for TGIF-TBC. More specifically, $T_K$ is derived as $T_K = \mathtt{encode}(L, D, B) = (2^D L) \oplus (0^{120} \,\|\, B)$ where $(L, D, B) \in \mathcal{L} \times \mathcal{D} \times \mathcal{B}$ is a part of input to ICE.

**Algorithm** $\mathsf{ICE}^{D,B}_{L,V}(M)$

1. $S \leftarrow 2^D V \oplus M$
2. $T_K \leftarrow \mathtt{encode}(L, D, B)$
3. $S \leftarrow E_{T_K}(S)$
4. $C \leftarrow 2^D V \oplus S$
5. **return** $C$

**Figure 2.4:** Definition of the TBC. ICE is either ICE1 or ICE2. Note that ICE1 fixes $V = 0^n$, hence effectively $S \leftarrow M$ (line 1) and $C \leftarrow S$ (line 4). Variables $L$ and $V$ are assumed to be derived from the corresponding KDF-ICE taking $(N, K)$, as a pre-processing.

**The Key Derivation Function** $\mathsf{KDF\text{-}ICE}(N, K)$**.** The KDF-ICE is used to generate $L$ and $V$ from the nonce $N$ and the master key $K$. Each ICE has its own KDF-ICE, defined as follows. For any KDF-ICE we have $|K| = |L| = 128$ and $|V| = n$.

1. KDF-ICE1: $|N| = 128$ and it is defined as $L \leftarrow G(E_K(N)), V \leftarrow 0^n$.
2. KDF-ICE2: $|N| = 128$ and it is defined as $L' \leftarrow E_K(N), V' \leftarrow E_{K \oplus 1}(L')$, and $L \leftarrow G(L'), V \leftarrow G(V')$.

When ICE is working inside TGIF, the corresponding KDF-ICE is performed only once as an initialization. For ICE1 or ICE2, the KDF-ICE involves one or two calls of $E$ and matrix multiplications by $G$ (see above).

For each input block, ICE applies doubling to the derived mask values. Since doubling is a sequential operation, computing $\mathsf{ICE}^{D+1,B}_{L,V}(M)$ after $\mathsf{ICE}^{D,B'}_{L,V}(M')$ is easy and does not need any additional memory, for any instance of ICE.

Note that, for convention, the function ICE (in Figure 2.4) is in fact defined as a core encryption function taking $(L, V, D, B)$ inside the TBC which takes $\mathcal{T} = \mathcal{N} \times \mathcal{D} \times \mathcal{B}$ as a tweak, where $\mathcal{N} = \{0,1\}^{nl}$ is the nonce space.

### 2.5.3 State Update Function

Let $G$ be an $n \times n$ binary matrix defined as an $n/8 \times n/8$ diagonal matrix of $8 \times 8$ binary sub-matrices:

$$G = \begin{pmatrix} G_s & 0 & 0 & \dots & 0 \\ 0 & G_s & 0 & \dots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & \dots & 0 & G_s & 0 \\ 0 & \dots & 0 & 0 & G_s \end{pmatrix}, \tag{2.2}$$

where 0 here represents the $8 \times 8$ zero matrix, and $G_s$ is an $8 \times 8$ binary matrix, defined as

$$G_s = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{2.3}$$

Alternatively, let $X \in \{0,1\}^n$, where $n$ is a multiple of 8, then the matrix-vector multiplication $G \cdot X$ can be represented as

$$G \cdot X = (G_s \cdot X[0], G_s \cdot X[1], G_s \cdot X[2], \ldots, G_s \cdot X[n/8-1]), \tag{2.4}$$

where

$$G_s \cdot X[i] = (X[i][1], X[i][2], X[i][3], X[i][4], X[i][5], X[i][6], X[i][7], X[i][7] \oplus X[i][0]) \tag{2.5}$$

for all $i \in [\![n/8]\!]_0$, such that $(X[0], \cdots, X[n/8-1]) \xleftarrow{8} X$ and $(X[i][0], ..X[i][7]) \xleftarrow{1} X[i]$, for all $i \in [\![n/8]\!]_0$.

The state update function $\rho : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n \times \{0,1\}^n$ and its inverse $\rho^{-1} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n \times \{0,1\}^n$ are defined as

$$\rho(S, M) = (S', C), \tag{2.6}$$

where $C = M \oplus G(S)$ and $S' = S \oplus M$. Similarly,

$$\rho^{-1}(S, C) = (S', M), \tag{2.7}$$

where $M = C \oplus G(S)$ and $S' = S \oplus M$. We note that we abuse the notation by writing $\rho^{-1}$ as this function is only the invert of $\rho$ according to its second parameter. For any $(S, M) \in \{0,1\}^n \times \{0,1\}^n$, if $\rho(S, M) = (S', C)$ holds then $\rho^{-1}(S, C) = (S', M)$. Besides, we remark that $\rho(S, 0^n) = (S, G(S))$ holds.

### 2.5.4 TGIF-N nonce-based AE mode

The specification of TGIF-N is shown in Figure 2.5. Figures 2.6 and 2.7 show encryption of TGIF-N. For completeness, the definition of $\rho$ is also included.

### 2.5.5 TGIF-M misuse-resistant AE mode

The specification of TGIF-M is shown in Figure 2.8. Figures 2.9 and 2.10 show encryption of TGIF-M. For completeness, the definition of $\rho$ is also included.

**Algorithm** $\mathsf{TGIF\text{-}N.Enc}_K(N, A, M)$

1. $(L, V) \leftarrow \mathsf{KDF\text{-}ICE}(N, K)$
2. $S \leftarrow 0^n$
3. $(A[1], \ldots, A[a]) \xleftarrow{n} A$
4. $(M[1], \ldots, M[m]) \xleftarrow{n} M$
5. **if** $|A[a]| < n$ **then** $w_A \leftarrow 13$ **else** 12
6. **if** $|M[m]| < n$ **then** $w_M \leftarrow 11$ **else** 10
7. $A[a] \leftarrow \mathtt{pad}_n(A[a])$
8. **for** $i = 1$ **to** $a - 1$
9. $\quad (S, \eta) \leftarrow \rho(S, A[i])$
10. $\quad S \leftarrow \mathsf{ICE}^{\vec{i},4}_{L,V}(S)$
11. **end for**
12. $(S, \eta) \leftarrow \rho(S, A[a])$
13. $S \leftarrow \mathsf{ICE}^{\vec{a},w_A}_{L,V}(S)$
14. **for** $i = 1$ **to** $m - 1$
15. $\quad (S, C[i]) \leftarrow \rho(S, M[i])$
16. $\quad S \leftarrow \mathsf{ICE}^{\overrightarrow{a+i},2}_{L,V}(S)$
17. **end for**
18. $M'[m] \leftarrow \mathtt{pad}_n(M[m])$
19. $(S, C'[m]) \leftarrow \rho(S, M'[m])$
20. $S \leftarrow \mathsf{ICE}^{\overrightarrow{a+m},w_M}_{L,V}(S)$
21. $C[m] \leftarrow \mathtt{lsb}_{|M[m]|}(C'[m])$
22. $(\eta, T) \leftarrow \rho(S, 0^n)$
23. $C \leftarrow C[1] \,\|\, C[2] \,\|\, \ldots \,\|\, C[m]$
24. **return** $(C, T)$

**Algorithm** $\mathsf{TGIF\text{-}N.Dec}_K(N, A, C, T)$

1. $(L, V) \leftarrow \mathsf{KDF\text{-}ICE}(N, K)$
2. $S \leftarrow 0^n$
3. $(A[1], \ldots, A[a]) \xleftarrow{n} A$
4. $(C[1], \ldots, C[m]) \xleftarrow{n} C$
5. **if** $|A[a]| < n$ **then** $w_A \leftarrow 13$ **else** 12
6. **if** $|C[m]| < n$ **then** $w_C \leftarrow 11$ **else** 10
7. $A[a] \leftarrow \mathtt{pad}_n(A[a])$
8. **for** $i = 1$ **to** $a - 1$
9. $\quad (S, \eta) \leftarrow \rho(S, A[i])$
10. $\quad S \leftarrow \mathsf{ICE}^{\vec{i},4}_{L,V}(S)$
11. **end for**
12. $(S, \eta) \leftarrow \rho(S, A[a])$
13. $S \leftarrow \mathsf{ICE}^{\vec{a},w_A}_{L,V}(S)$
14. **for** $i = 1$ **to** $m - 1$
15. $\quad (S, M[i]) \leftarrow \rho^{-1}(S, C[i])$
16. $\quad S \leftarrow \mathsf{ICE}^{\overrightarrow{a+i},2}_{L,V}(S)$
17. **end for**
18. $\widetilde{S} \leftarrow (0^{|C[m]|} \,\|\, \mathtt{msb}_{n-|C[m]|}(G(S)))$
19. $C'[m] \leftarrow \mathtt{pad}_n(C[m]) \oplus \widetilde{S}$
20. $(S, M'[m]) \leftarrow \rho^{-1}(S, C'[m])$
21. $M[m] \leftarrow \mathtt{lsb}_{|C[m]|}(M'[m])$
22. $S \leftarrow \mathsf{ICE}^{\overrightarrow{a+m},w_C}_{L,V}(S)$
23. $(\eta, T^*) \leftarrow \rho(S, 0^n)$
24. $M \leftarrow M[1] \,\|\, M[2] \,\|\, \ldots \,\|\, M[m]$
25. **if** $T^* = T$ **then return** $M$ **else** $\perp$

**Algorithm** $\rho(S, M)$

1. $C \leftarrow M \oplus G(S)$
2. $S' \leftarrow S \oplus M$
3. **return** $(S', C)$

**Algorithm** $\rho^{-1}(S, C)$

1. $M \leftarrow C \oplus G(S)$
2. $S' \leftarrow S \oplus M$
3. **return** $(S', M)$

**Figure 2.5:** TGIF-N using ICE. Lines of [**if (statement) then** $X \leftarrow x$ **else** $x'$] are shorthand for [**if (statement) then** $X \leftarrow x$ **else** $X \leftarrow x'$]. The dummy variable $\eta$ is always discarded. TGIF-N1 is used as a working example. For TGIF-N2 version, the values of the bit $b_6$ in the domain separation need to be set to 1, alongside with using the appropriate ICE and KDF-ICE variants.

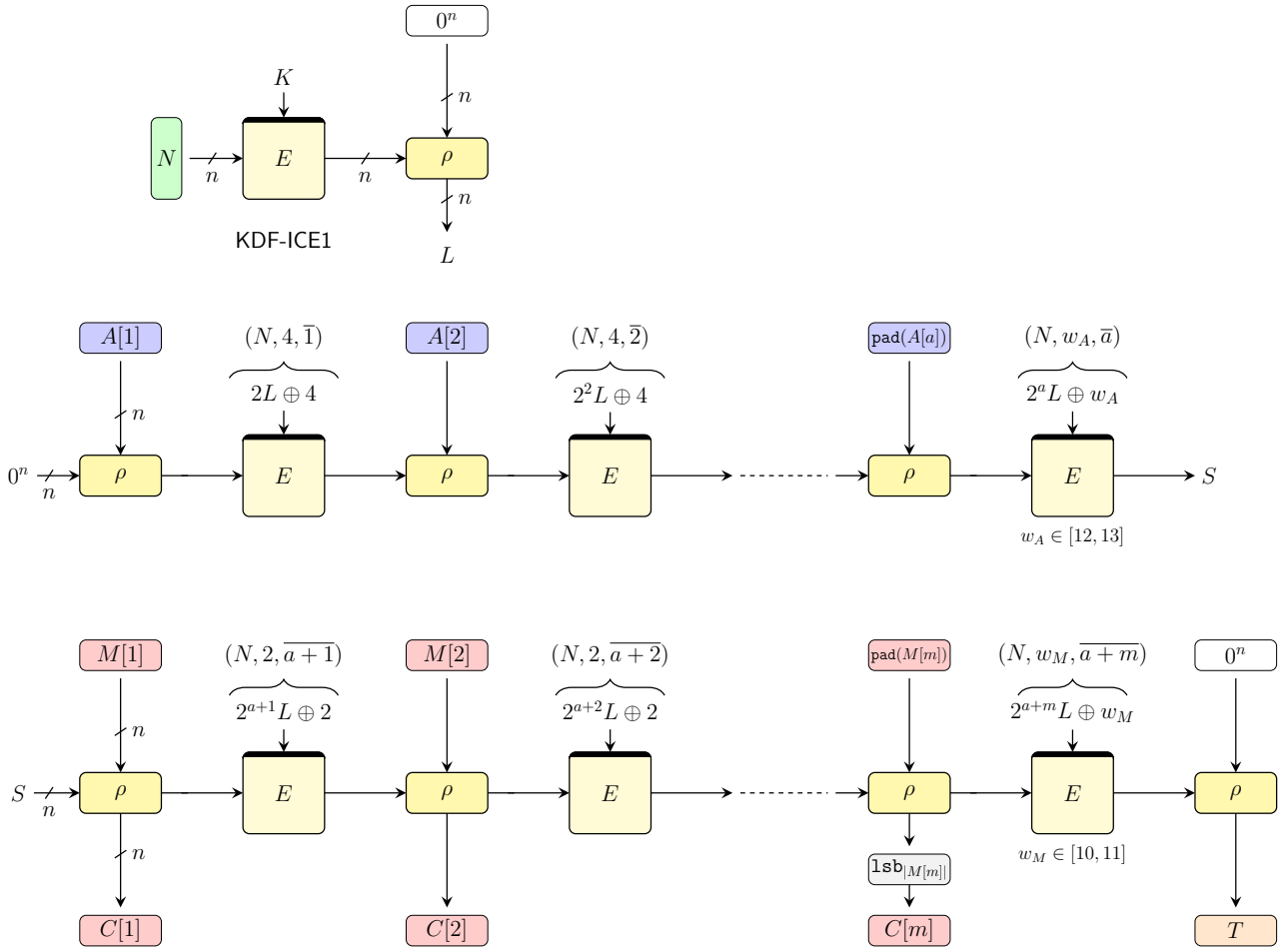


**Figure 2.6:** TGIF-N with ICE1 (TGIF-N1). (Top) Key derivation. (Middle) Processing of AD (Bottom) Encryption. The domain separation $B$ being of 8 bits only, $\oplus B$ is to be interpreted as $\oplus\, 0^{120}||B$.

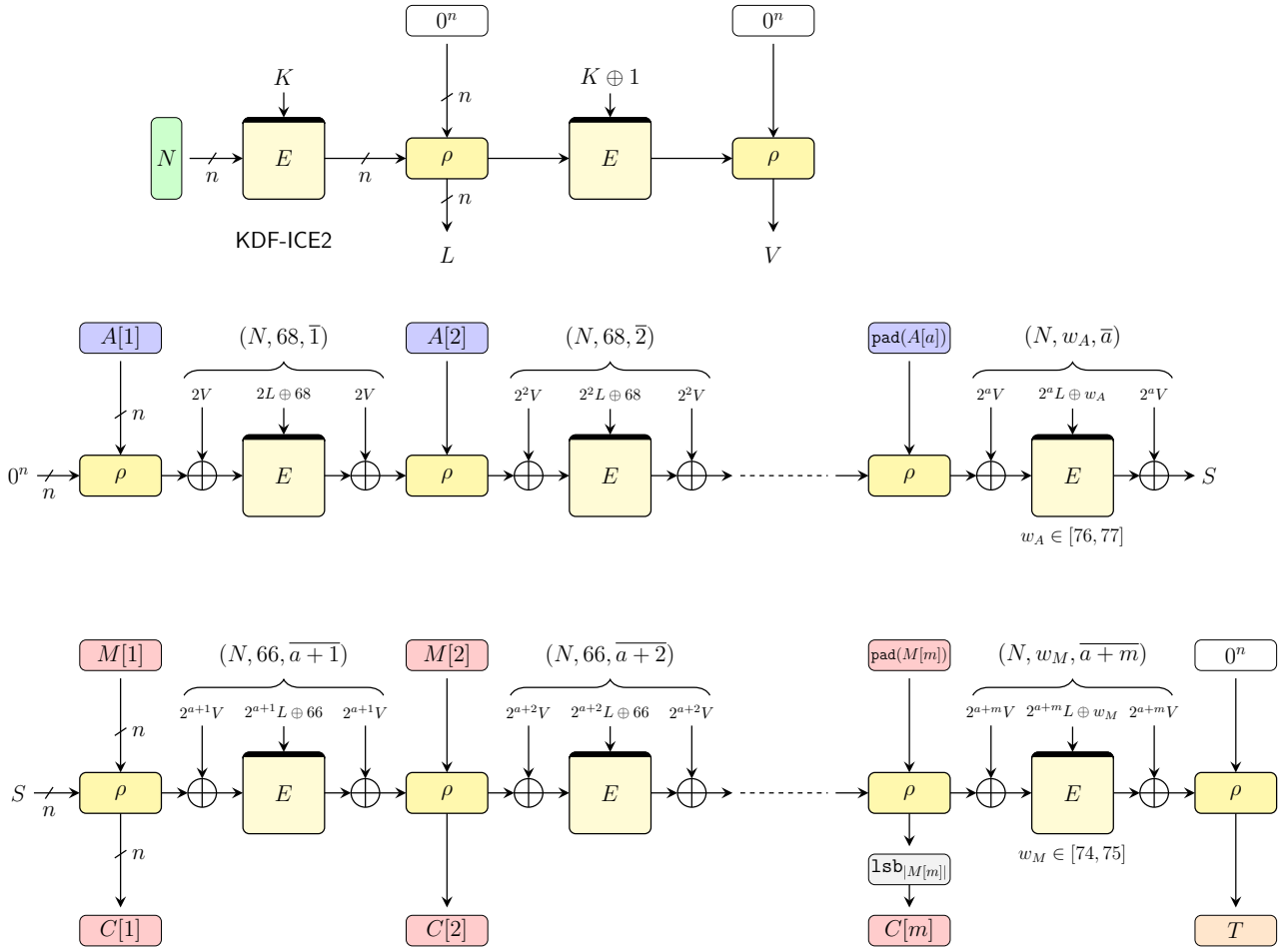


**Figure 2.7:** TGIF-N with ICE2 (TGIF-N2). (Top) Key derivation. (Middle) Processing of AD (Bottom) Encryption. The domain separation $B$ being of 8 bits only, $\oplus B$ is to be interpreted as $\oplus\, 0^{120}||B$.

**Algorithm** $\mathsf{TGIF\text{-}M.Enc}_K(N, A, M)$

1. $(L, V) \leftarrow \mathsf{KDF\text{-}ICE}(N, K)$
2. $S \leftarrow 0^n$
3. $(A[1], \ldots, A[a]) \xleftarrow{n} A$
4. $(M[1], \ldots, M[m]) \xleftarrow{n} M$
5. **if** $|A[a]| < n$ **then** $w_A \leftarrow 45$ **else** 44
6. **if** $|M[m]| < n$ **then** $w_M \leftarrow 47$ **else** 46
7. $A[a] \leftarrow \mathtt{pad}_n(A[a])$
8. **for** $i = 1$ **to** $a - 1$
9. $\quad (S, \eta) \leftarrow \rho(S, A[i])$
10. $\quad S \leftarrow \mathsf{ICE}_{L,V}^{\bar{i},36}(S)$
11. **end for**
12. $(S, \eta) \leftarrow \rho(S, A[a])$
13. $S \leftarrow \mathsf{ICE}_{L,V}^{\bar{a},w_A}(S)$
14. **for** $i = 1$ **to** $m - 1$
15. $\quad (S, \eta) \leftarrow \rho(S, M[i])$
16. $\quad S \leftarrow \mathsf{ICE}_{L,V}^{\overline{a+i},38}(S)$
17. **end for**
18. $M'[m] \leftarrow \mathtt{pad}_n(M[m])$
19. $(S, \eta) \leftarrow \rho(S, M'[m])$
20. $S \leftarrow \mathsf{ICE}_{L,V}^{\overline{a+m},w_M}(S)$
21. $(\eta, T) \leftarrow \rho(S, 0^n)$
22. **if** $M = \epsilon$ **then return** $(\epsilon, T)$
23. $S \leftarrow T$
24. **for** $i = 1$ **to** $m - 1$
25. $\quad S \leftarrow \mathsf{ICE}_{L,V}^{\overline{i-1},34}(S)$
26. $\quad (S, C[i]) \leftarrow \rho(S, M[i])$
27. **end for**
28. $S \leftarrow \mathsf{ICE}_{L,V}^{\overline{m-1},34}(S)$
29. $(\eta, C'[m]) \leftarrow \rho(S, M'[m])$
30. $C[m] \leftarrow \mathtt{lsb}_{|M[m]|}(C'[m])$
31. $C \leftarrow C[1] \,\|\, C[2] \,\|\, \ldots \,\|\, C[m]$
32. **return** $(C, T)$

**Algorithm** $\mathsf{TGIF\text{-}M.Dec}_K(N, A, C, T)$

1. $(L, V) \leftarrow \mathsf{KDF\text{-}ICE}(N, K)$
2. **if** $C = \epsilon$ **then** $M \leftarrow \epsilon$
3. **else**
4. $\quad S \leftarrow T$
5. $\quad (C[1], \ldots, C[m]) \xleftarrow{n} C$
6. $\quad z \leftarrow |C[m]|$
7. $\quad C[m] \leftarrow \mathtt{pad}_n(C[m])$
8. $\quad$ **for** $i = 1$ **to** $m$
9. $\quad\quad S \leftarrow \mathsf{ICE}_{L,V}^{\overline{i-1},34}(S)$
10. $\quad\quad (S, M[i]) \leftarrow \rho^{-1}(S, C[i])$
11. $\quad$ **end for**
12. $\quad M[m] \leftarrow \mathtt{lsb}_z(M[m])$
13. $\quad M \leftarrow M[1] \,\|\, \ldots \,\|\, M[m]$
14. $S \leftarrow 0^n$
15. $(A[1], \ldots, A[a]) \xleftarrow{n} A$
16. **if** $|A[a]| < n$ **then** $w_A \leftarrow 45$ **else** 44
17. **if** $|M[m]| < n$ **then** $w_M \leftarrow 47$ **else** 46
18. $A[a] \leftarrow \mathtt{pad}_n(A[a])$
19. **for** $i = 1$ **to** $a - 1$
20. $\quad (S, \eta) \leftarrow \rho(S, A[i])$
21. $\quad S \leftarrow \mathsf{ICE}_{L,V}^{\bar{i},36}(S)$
22. **end for**
23. $(S, \eta) \leftarrow \rho(S, A[a])$
24. $S \leftarrow \mathsf{ICE}_{L,V}^{\bar{a},w_A}(S)$
25. **for** $i = 1$ **to** $m - 1$
26. $\quad (S, \eta) \leftarrow \rho(S, M[i])$
27. $\quad S \leftarrow \mathsf{ICE}_{L,V}^{\overline{a+i},38}(S)$
28. **end for**
29. $M'[m] \leftarrow \mathtt{pad}_n(M[m])$
30. $(S, \eta) \leftarrow \rho(S, M'[m])$
31. $S \leftarrow \mathsf{ICE}_{L,V}^{\overline{a+m},w_M}(S)$
32. $(\eta, T^*) \leftarrow \rho(S, 0^n)$
33. **if** $T^* = T$ **then return** $M$ **else** $\perp$

**Algorithm** $\rho(S, M)$

1. $C \leftarrow M \oplus G(S)$
2. $S' \leftarrow S \oplus M$
3. **return** $(S', C)$

**Algorithm** $\rho^{-1}(S, C)$

1. $M \leftarrow C \oplus G(S)$
2. $S' \leftarrow S \oplus M$
3. **return** $(S', M)$

**Figure 2.8:** TGIF-M using ICE. TGIF-M1 is used as a working example. For TGIF-M2 version, the values of the bit $b_6$ in the domain separation need to be set to 1, alongside with using the appropriate ICE and KDF-ICE variants.

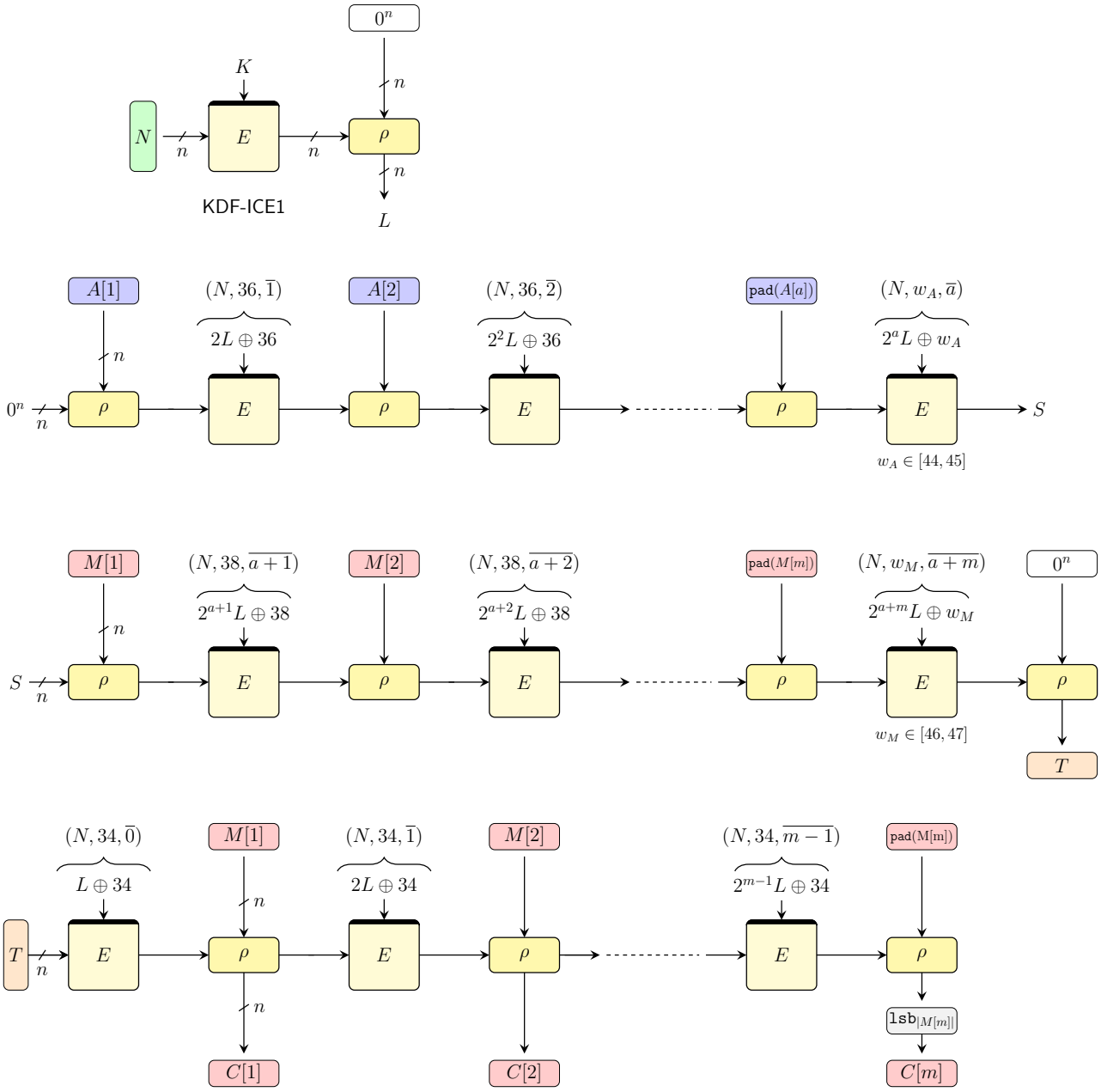


**Figure 2.9:** TGIF-M with ICE1 (TGIF-M1). (Top) Key derivation. (Middle-Top) Processing of AD (Middle-Bottom) Processing of M authentication (Bottom) Encryption. The domain separation $B$ being of 8 bits only, $\oplus B$ is to be interpreted as $\oplus\, 0^{120}||B$.

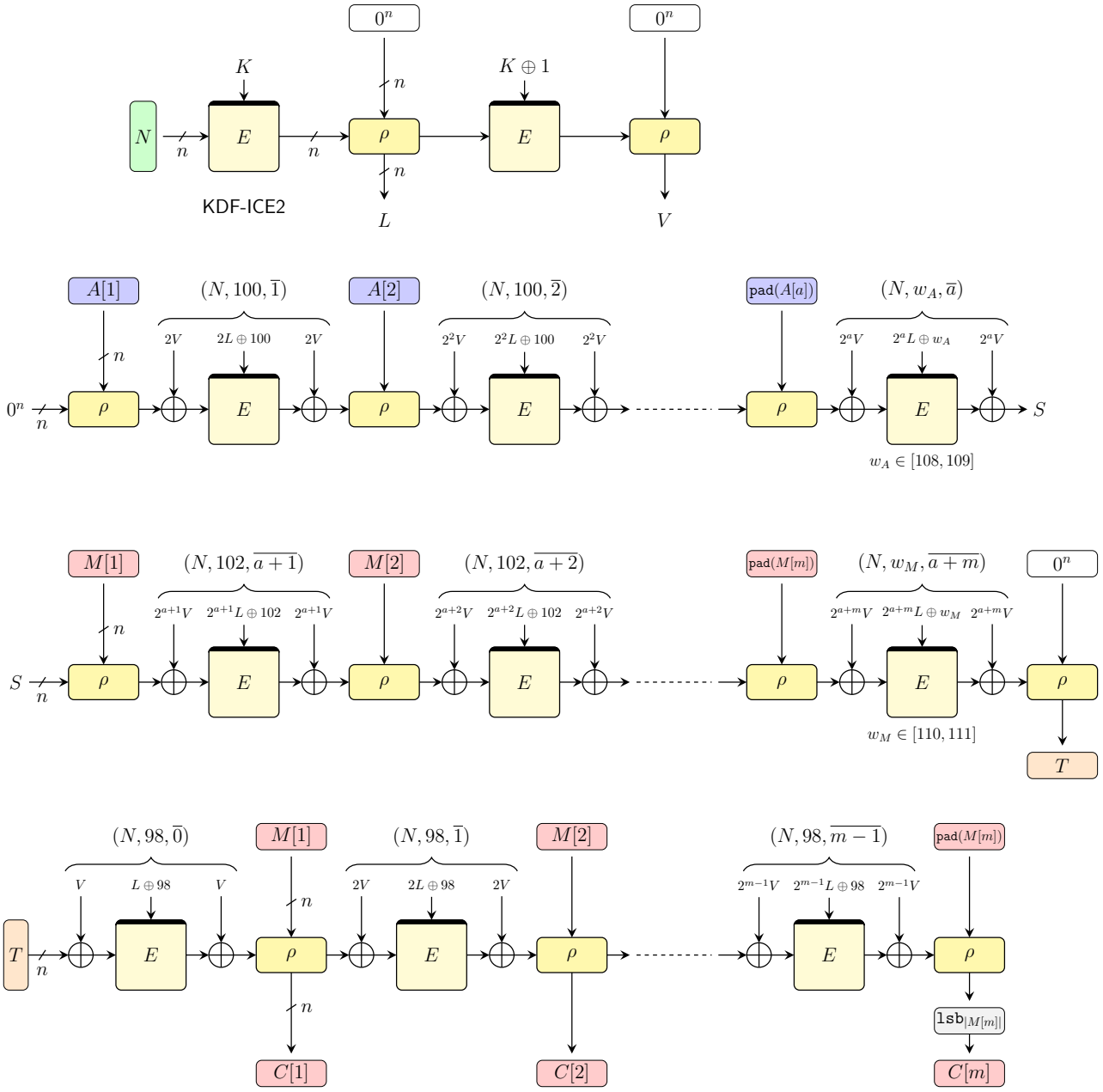


**Figure 2.10:** TGIF-M with ICE2 (TGIF-M2). (Top) Key derivation. (Middle-Top) Processing of AD (Middle-Bottom) Processing of M authentication (Bottom) Encryption. The domain separation $B$ being of 8 bits only, $\oplus B$ is to be interpreted as $\oplus\, 0^{120}||B$.

# Security Claims

**Attack Models.** We consider two models of adversaries: nonce-respecting (NR) and nonce-misusing (NM)[1]. In the former model, nonce values in encryption queries (the tuples $(N, A, M)$) may be chosen by the adversary but they must be distinct. In the latter, nonce values in encryption queries can repeat. Basically, an NM adversary can arbitrarily repeat a nonce, hence even using the same nonce for all queries is possible. We can further specify NM by the distribution of a nonce, such as the maximum number of repetition of a nonce in the encryption queries. For both models, adversaries can use any nonce values in decryption queries (the tuples $(N, A, C, T)$): it can collide with a nonce in an encryption query or with other decryption queries.

**Security Claims.** Our security claims are summarized in Table 3.1. The variables in the table denote the required workload, in terms of online and offline query complexities (corresponding to data complexity and time complexity), of an adversary to break the cipher, in logarithm base 2. In more detail, an integer $x$ in the table means an attack possibly breaks the scheme with online query complexity $Q_{\text{online}}$ and offline query complexity $Q_{\text{offline}}$ if $\max\{Q_{\text{online}}, Q_{\text{offline}}\} = 2^x$. For simplicity, small constant factors, which are determined from the concrete security bounds, are neglected in these tables. A more detailed analysis is given in Section 4.

We claim these numbers under the Ideal-Cipher Model (ICM), that is, the model that assumes TGIF-TBC is sampled uniformly over all the ciphers (see Section 4 for the definition). Intuitively, this corresponds to model that TGIF-TBC behaves ideally, though ICM cannot be instantiated (as a key of $E$ is also a part of queries to ICM, and outputs must be random). ICM has been acknowledged as a meaningful security proof model, especially in the field of hash function constructions.

For TGIF-N1, Table 3.1 shows $n/2$-bit security for both privacy and authenticity against NR adversary. For TGIF-N2, Table 3.1 shows full $n$-bit security for both privacy and authenticity against NR adversary. For TGIF-M1, Table 3.1 shows $n/2$-bit security for both privacy and authenticity against NR and NM adversaries. For TGIF-M2, Table 3.1 shows $n$-bit security for both privacy and authenticity against NR adversary and in addition, $n/2$-bit security for both privacy and authenticity against NM adversary. The $n/2$-bit security assumes that the NM adversary has full control over the nonce, but in practice, the nonce repetition can happen accidentally, and it is conceivable that the nonce is repeated only a few times. As we present in Section 4, the security bounds of TGIF-M2 show the notable property of graceful security degradation with respect to the number of nonce repetition [22]. This property is similar to SCT, and if the number of nonce repetition is limited, the actual security bound is close to the full $n$-bit security.

[1] Also known as Nonce Repeating or Nonce Ignoring. We chose "Nonce Misuse" for notational convenience of using acronyms, NR for nonce-respecting and NM for nonce-misuse.

**Table 3.1:** Security claims of TGIF. NR denotes Nonce-Respecting adversary and NM denotes Nonce-Misusing adversary.

| Parameter | NR-Priv | NR-Auth | NM-Priv | NM-Auth |
|---|---|---|---|---|
| TGIF-N1 | 64 | 64 | – | – |
| TGIF-N2 | 128 | 128 | – | – |
| TGIF-M1 | 64 | 64 | 64 | 64 |
| TGIF-M2 | 128 | 128 | $64 \sim 128$ | $64 \sim 128$ |

# Security Analysis

## 4.1 Security of the mode

### 4.1.1 Security Notions

**Security Notions for NAE.** We consider the standard security notions for nonce-based AE [3, 4, 25]. Let $\Pi$ denote an NAE scheme consisting of an encryption procedure $\Pi.\mathcal{E}_K$ and a decryption procedure $\Pi.\mathcal{D}_K$, for secret key $K$ uniform over set $\mathcal{K}$ (denoted as $K \xleftarrow{\$} \mathcal{K}$). For plaintext $M$ with nonce $N$ and associated data $A$, $\Pi.\mathcal{E}_K$ takes $(N, A, M)$ and returns ciphertext $C$ (typically $|C| = |M|$) and tag $T$. For decryption, $\Pi.\mathcal{D}_K$ takes $(N, A, C, T)$ and returns a decrypted plaintext $M$ if authentication check is successful, and otherwise an error symbol, $\perp$. We assume $\Pi$ is based on the ideal block cipher $E : \mathcal{K} \times \mathcal{M} \to \mathcal{M}$, which is uniformly distributed over all block ciphers of key space $\mathcal{K}$ and message space $\mathcal{M}$, and we allow the adversary to query $E$ while attacking $\Pi$. Specifically, the adversary can query $Y \leftarrow E(K', X)$ for any $(K', X) \in \mathcal{K} \times \mathcal{M}$ or $X \leftarrow E^{-1}(K', Y)$ for any $(K', Y) \in \mathcal{K} \times \mathcal{M}$. We remark that $K'$ here is a part of query and not the secret key $K \xleftarrow{\$} \mathcal{K}$. Such a query is called an offline query or a primitive query. In contrast, a query to $\Pi.\mathcal{E}_K$ or $\Pi.\mathcal{D}_K$ is called an online query or a construction query.

The privacy notion is the indistinguishability of encryption oracle $\Pi.\mathcal{E}_K$ from the random-bit oracle \$ which returns random $|M| + \tau$ bits for any query $(N, A, M)$, with access to the ideal cipher $E$ for both worlds (\$ oracle and $E$ oracle are independent). The adversary is assumed to be nonce-respecting. We define the privacy advantage as

$$\mathbf{Adv}^{\mathtt{priv}}_{\Pi}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\Pi.\mathcal{E}_K(\cdot,\cdot,\cdot),(E,E^{-1})} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot,\cdot),(E,E^{-1})} \Rightarrow 1\right]$$

which measures the hardness of breaking the privacy notion for $\mathcal{A}$.

The authenticity notion is the probability of successful forgery via queries to $\Pi.\mathcal{E}_K$ and $\Pi.\mathcal{D}_K$ oracles. As in the case of the privacy notion, the adversary has access to $E$. We define the authenticity advantage as

$$\mathbf{Adv}^{\mathtt{auth}}_{\Pi}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\Pi.\mathcal{E}_K(\cdot,\cdot,\cdot),\Pi.\mathcal{D}_K(\cdot,\cdot,\cdot,\cdot),(E,E^{-1})} \text{ forges } \right],$$

where $\mathcal{A}$ forges if it receives a value $M' \neq \perp$ from $\Pi.\mathcal{D}_K$. Here, to prevent trivial wins, if $(C, T) \leftarrow \Pi.\mathcal{E}_K(N, A, M)$ is obtained earlier, $\mathcal{A}$ cannot query $(N, A, C, T)$ to $\Pi.\mathcal{D}_K$. The adversary is assumed to be nonce-respecting for encryption queries.

**Security Notions for MRAE.** We adopt the security notions of MRAE following the same security definitions as above, with the exception that the adversary can now repeat nonces. We write the corresponding privacy advantage as

$$\mathbf{Adv}^{\mathtt{nm\text{-}priv}}_{\Pi}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\Pi.\mathcal{E}_K(\cdot,\cdot,\cdot),(E,E^{-1})} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot,\cdot),(E,E^{-1})} \Rightarrow 1\right],$$

and the authenticity advantage as

$$\mathbf{Adv}^{\mathtt{nm\text{-}auth}}_{\Pi}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\Pi.\mathcal{E}_K(\cdot,\cdot,\cdot),\Pi.\mathcal{D}_K(\cdot,\cdot,\cdot,\cdot),(E,E^{-1})} \text{ forges}\right].$$

We note that while adversaries can repeat nonces, we without loss of generality assume that they do not repeat the same query. See also [26] for reference.

### 4.1.2 Security of TGIF-N

For $A \in \{0,1\}^*$, we say $A$ has $a$ AD blocks if $|A|_n = a$. Similarly for plaintext $M \in \{0,1\}^*$ we say $M$ has $m$ message blocks if $|M|_n = m$. The same holds for the ciphertext $C$. For encryption query $(N, A, M)$ or decryption query $(N, A, C, T)$ of $a$ AD blocks and $m$ message blocks, the number of total TBC calls is at most $a + m$, which is called the number of *effective blocks* of a query.

Let $\mathcal{A}$ be a nonce-respecting adversary against TGIF-N using $q_c$ encryption (online/construction) queries and $q_p$ offline/primitive queries with total number of effective blocks in encryption queries $\sigma_{\text{priv}}$. We have the following privacy bounds:

$$\mathbf{Adv}^{\mathtt{priv}}_{\mathsf{TGIF\text{-}N1}}(\mathcal{A}) \le \frac{9\sigma^2_{\text{priv}} + 4\sigma_{\text{priv}} \cdot q_p}{2^n} + \frac{2q_p}{2^n}, \tag{4.1}$$

$$\mathbf{Adv}^{\mathtt{priv}}_{\mathsf{TGIF\text{-}N2}}(\mathcal{A}) \le \frac{9\sigma^2_{\text{priv}} + 4\sigma_{\text{priv}} \cdot q_p}{2^{2n}} + \frac{2q_p}{2^n}. \tag{4.2}$$

For authenticity bounds, let $\mathcal{B}$ be a nonce-respecting adversary using $q_c$ encryption queries and $q_d$ decryption queries (both are online/construction queries), with total number of effective blocks for encryption and decryption queries $\sigma_{\text{auth}}$, and $q_p$ offline/primitive queries. Also we define $\ell$ as the maximum effective block length of a plaintext among $q_c$ encryption queries. Then we have

$$\mathbf{Adv}^{\mathtt{auth}}_{\mathsf{TGIF\text{-}N1}}(\mathcal{A}) \le \frac{9\sigma^2_{\text{auth}} + 4\sigma_{\text{auth}} \cdot q_p}{2^n} + \frac{2q_p}{2^n} + \frac{2\ell q_d}{2^n} + \frac{2q_d}{2^\tau}, \tag{4.3}$$

$$\mathbf{Adv}^{\mathtt{auth}}_{\mathsf{TGIF\text{-}N2}}(\mathcal{A}) \le \frac{9\sigma^2_{\text{auth}} + 4\sigma_{\text{auth}} \cdot q_p}{2^{2n}} + \frac{2q_p}{2^n} + \frac{2\ell q_d}{2^n} + \frac{2q_d}{2^\tau}. \tag{4.4}$$

Note that tag length $\tau$ is set to $n$ for all members of TGIF-N, however, if $1 \le \tau < n$ (which is not a part of our submission), it still maintains $n$-bit privacy and $\tau$-bit authenticity. While the term $\frac{2\ell q_d}{2^n}$ is present, this term is very likely not tight and can be improved to $\sigma_{\text{dec}}/2^n$ with total effective blocks in decryption queries $\sigma_{\text{dec}}$.

The security of TGIF-N crucially relies on the $n \times n$ matrix $G$ defined over GF(2). Let $G^{(i)}$ be an $n \times n$ matrix that is equal to $G$ except the $(i+1)$-st to $n$-th rows, which are set to all zero. Here, $G^{(0)}$ is the zero matrix and $G^{(n)} = G$, and for $X \in \{0,1\}^n$, $G^{(i)}(X) = \mathtt{lsb}_i(G(X))\|0^{n-i}$ for all $i = 0, 8, 16, \ldots, n$; note that all variables are byte strings, and $\mathtt{lsb}_i(X)$ is the leftmost $i/8$ bytes (Section 2). Let $I$ denote the $n \times n$ identity matrix. We say $G$ is sound if (1) $G$ is regular and (2) $G^{(i)} + I$ is regular for all $i = 8, 16, \ldots, n$. The above security bounds hold as long as $G$ is sound. The proofs are similar to those for iCOFB [8]. We have verified the soundness of our $G$, for a range of $n$ including $n = 64$ and $n = 128$, by a computer program.

### 4.1.3 Security of TGIF-M

For encryption query $(N, A, M)$ or decryption query $(N, A, C, T)$ of $a$ AD blocks and $m$ message blocks, the number of total TBC calls is at most $a + 2m$, which is called the number of *effective blocks* of a query.

Let $\mathcal{A}$ be an adversary against TGIF-N using $q_c$ encryption (online/construction) queries and $q_p$ offline/primitive queries with total number of effective blocks in encryption queries $\sigma_{\text{priv}}$. In the

NR case, we have the following privacy bounds:

$$\mathbf{Adv}^{\mathtt{priv}}_{\mathsf{TGIF\text{-}M1}}(\mathcal{A}) \leq \frac{9\sigma^2_{\mathrm{priv}} + 4\sigma_{\mathrm{priv}} \cdot q_p}{2^n} + \frac{2q_p}{2^n}, \tag{4.5}$$

$$\mathbf{Adv}^{\mathtt{priv}}_{\mathsf{TGIF\text{-}M2}}(\mathcal{A}) \leq \frac{9\sigma^2_{\mathrm{priv}} + 4\sigma_{\mathrm{priv}} \cdot q_p}{2^{2n}} + \frac{2q_p}{2^n}. \tag{4.6}$$

In the NM case, we have the following privacy bounds:

$$\mathbf{Adv}^{\mathtt{nm\text{-}priv}}_{\mathsf{TGIF\text{-}M1}}(\mathcal{A}) \leq \frac{9\sigma^2_{\mathrm{priv}} + 4\sigma_{\mathrm{priv}} \cdot q_p}{2^n} + \frac{2q_p}{2^n} + \frac{4r\sigma_{\mathrm{priv}}}{2^n}, \tag{4.7}$$

$$\mathbf{Adv}^{\mathtt{nm\text{-}priv}}_{\mathsf{TGIF\text{-}M2}}(\mathcal{A}) \leq \frac{9\sigma^2_{\mathrm{priv}} + 4\sigma_{\mathrm{priv}} \cdot q_p}{2^{2n}} + \frac{2q_p}{2^n} + \frac{4r\sigma_{\mathrm{priv}}}{2^n}. \tag{4.8}$$

Here, the adversary can repeat a nonce at most $r$ times.

For authenticity bounds, let $\mathcal{B}$ be an adversary using $q_c$ encryption queries and $q_d$ decryption queries (both are online/construction queries), with total number of effective blocks for encryption and decryption queries $\sigma_{\mathrm{auth}}$, and $q_p$ offline/primitive queries. Also we define $\ell$ as the maximum effective block length among all the encryption and decryption queries. Then in the NR case, we have

$$\mathbf{Adv}^{\mathtt{auth}}_{\mathsf{TGIF\text{-}M1}}(\mathcal{A}) \leq \frac{9\sigma^2_{\mathrm{auth}} + 4\sigma_{\mathrm{auth}} \cdot q_p}{2^n} + \frac{2q_p}{2^n} + \frac{2\ell q_d}{2^n}, \tag{4.9}$$

$$\mathbf{Adv}^{\mathtt{auth}}_{\mathsf{TGIF\text{-}M2}}(\mathcal{A}) \leq \frac{9\sigma^2_{\mathrm{auth}} + 4\sigma_{\mathrm{auth}} \cdot q_p}{2^{2n}} + \frac{2q_p}{2^n} + \frac{2\ell q_d}{2^n}. \tag{4.10}$$

In the NM case, we have

$$\mathbf{Adv}^{\mathtt{nm\text{-}auth}}_{\mathsf{TGIF\text{-}M1}}(\mathcal{A}) \leq \frac{9\sigma^2_{\mathrm{auth}} + 4\sigma_{\mathrm{auth}} \cdot q_p}{2^n} + \frac{2q_p}{2^n} + \frac{2r\ell q_d}{2^n}, \tag{4.11}$$

$$\mathbf{Adv}^{\mathtt{nm\text{-}auth}}_{\mathsf{TGIF\text{-}M2}}(\mathcal{A}) \leq \frac{9\sigma^2_{\mathrm{auth}} + 4\sigma_{\mathrm{auth}} \cdot q_p}{2^{2n}} + \frac{2q_p}{2^n} + \frac{2r\ell q_d}{2^n}. \tag{4.12}$$

The term $O(\frac{r\ell q_d}{2^n})$ is likely not tight and can be improved to $O(r\sigma_{\mathrm{dec}}/2^n)$ with total effective blocks in decryption queries $\sigma_{\mathrm{dec}}$.

## 4.2 Security of TGIF-TBC

To be added soon.

# Features

The primary goal of TGIF is to provide a lightweight, yet highly-secure, highly-efficient AE based on a new TBC named TGIF-TBC. TGIF has a number of desirable features. Below we detail some representative ones:

- **Performance of TBC.** TGIF-TBC is an extremely efficient TBC, both in terms of area and throughput, on a broad range of devices, from very constrained hardware to high-end software platforms. It has the very interesting property that it can either been seen as a classical SPN cipher (which makes it very easy to analyse its security and derive bounds for differential/linear cryptanalysis) or as a bitslice-oriented cipher that allows very efficient constant time bitslice implementations, even without parallel primitive calls. Its most efficient implementation is straightforward and can be easily adapted to various architectures and to enable area/throughput trade-offs. TGIF-TBC is even more efficient than the 128-bit version of GIFT (one of the most lightweight block cipher to date), while providing stronger security guarantees. It is as of today one of the 128-bit primitive that uses the smallest number of basic boolean operations. Moreover, it has the special property that its key schedule naturally brings back the key state to its original input value at the end of the cipher computation, which will allow the designer to save area in the mode.

**Table 5.1:** Hardware performances of round-based implementations of GIFT, SKINNY, SIMON and our new cipher TGIF-TBC, synthesized with STM 90nm Standard cell library.

| | Area (GE) | Delay (ns) | Cycles | $TP_{MAX}$ (MBit/s) | Power ($\mu$W) (@10MHz) | Energy (pJ) |
|---|---|---|---|---|---|---|
| TGIF-TBC | | | 72 | | | |
| GIFT-128 | 1997 | 1.85 | 41 | 1729.7 | 116.6 | 478.1 |
| SKINNY-128-128 | 2104 | 1.85 | 41 | 1729.7 | 132.5 | 543.3 |
| SIMON 128/128 | 2064 | 1.87 | 69 | 1006.6 | 105.6 | 728.6 |

- **Performance of Mode.** TGIF-N mode is efficient: it encrypts $n$-bit block by just one call of $n$-bit-block primitive. Besides it is smaller than ΘCB3 in that it does not need an additional state beyond the internal TBC and it requires a smaller TBC (smaller tweakey). Although, TGIF is serial in nature, *i.e.*, not parallelizable, it was shown during the CAESAR competition that parallelizability does not lead to significant performance gains in hardware performance [10, 16, 18]. Moreover, parallelizability is not considered crucial in lightweight applications, so it is a small price for a simple, small and fast design.
- **Security of TBC.** The dual nature of TGIF-TBC (SPN/bitslice) allows us to study TGIF-TBC as a classical SPN cipher, with enough structure so that automated tools (SAT, MILP, etc.) can easily obtain bounds on the number of active boxes in a differential or linear attack.
- **Security proofs for the mode.** Both TGIF-N and TGIF-M have provable security in the

**Table 5.2:** Total number of operations and theoretical performance of GIFT and various lightweight block ciphers. N denotes a NOR gate, A denotes a AND gate, X denotes a XOR gate.

| Cipher | nb. of rds | gate cost (per bit per round) | | | nb. of op. w/o key sch. | nb. of op. w/ key sch. | round-based impl. area |
|---|---|---|---|---|---|---|---|
| | | int. cipher | key sch. | total | | | |
| TGIF-TBC | 72 | 0.5 N<br>0.87 X | 0.5 X | 0.5 N<br>1.37 X | $1.37 \times 72$<br>$= 99$ | $1.87 \times 72$<br>$=$ **135** | $0.5 + 2.67 \times 1.37$<br>$=$ **4.17** |
| GIFT<br>-128-128 | 40 | 1 N<br>2 X | | 1 N<br>2 X | $3.0 \times 40$<br>$= 120$ | $3.0 \times 40$<br>$=$ **120** | $1 + 2.67 \times 2$<br>$=$ **6.34** |
| SKINNY<br>-128-128 | 40 | 1 N<br>2.25 X | | 1 N<br>2.25 X | $3.25 \times 40$<br>$= 130$ | $3.25 \times 40$<br>$=$ **130** | $1 + 2.67 \times 2.25$<br>$=$ **7.01** |
| SIMON<br>-128/128 | 68 | 0.5 A<br>1.5 X | 1 X | 0.5 A<br>2.5 X | $2 \times 68$<br>$= 136$ | $3 \times 68$<br>$=$ **204** | $0.67 + 2.67 \times 2.5$<br>$=$ **7.34** |
| AES<br>-128 | 10 | 4.25 A<br>16 X | 1.06 A<br>3.5 X | 5.31 A<br>19.5 X | $20.25 \times 10$<br>$= 202.5$ | $24.81 \times 10$<br>$=$ **248.1** | $7.06 + 2.67 \times 19.5$<br>$=$ **59.12** |

Ideal-Cipher Model (ICM), where TGIF-TBC is modeled as the ideal-cipher. This is very important for high security confidence of TGIF and allows us to rely on the security of TGIF to that of TGIF-TBC, which has been extensively studied since the proposal in 2016. As described above, TGIF-TBC has strong security even under the related-key related-tweakey model, which is relevant to TGIF. Our provable security results are relying on the model where TGIF-TBC behaves as the ideal-cipher. It tells us that unless we find a structural weakness of TGIF-TBC, TGIF will be secure, as in the same way of reasoning for (public) permutation-based schemes, such as Sponge.

- **Beyond-birthday-bound security (TGIF-N2 and TGIF-M2).** The security bounds of TGIF-N2 shown in Section 4 are comparable to the state-of-the-art TBC modes of operation, namely ΘCB3 for NAE and SCT for MRAE. In particular, TGIF-N2 and TGIF-M2 (under NR adversary) achieve beyond-birthday-bound (BBB) security with respect to the block length. This level of security is much stronger than the up-to-birthday-bound, $n/2$-bit security achieved by conventional block cipher modes using $n$-bit block ciphers, *e.g.* GCM. We note that the above comparison ignores the fact that ΘCB3 and SCT use a dedicated TBC while ours use an (ideal) block cipher. This is simply because of the lack of BBB-secure NAE/MRAE schemes based on the ideal-cipher: combining known ICM-to-TBC results [14, 20, 27] and TBC-based modes would be possible, however, the performance will be quite poor unless the tweak-dependent key derivation is very carefully considered, which is exactly the point we carefully elaborated in our design.
- **Misuse resistance.** TGIF-M is an MRAE mode which is secure against misuse (repeat) of nonces in encryption queries. More formally, it provides the best-possible security against nonce repeat in that ciphertexts do not give any information as long as the uniqueness of the input tuple $(N, A, M)$ is maintained. In contrast to this, popular nonce-based AE modes are often vulnerable against nonce repeat, even one repetition can be significant. For example, the famous nonce repeat attack against GCM [11, 15] reveals its authentication key. While TGIF-N is nonce-based AE (NAE) secure against standard nonce-respecting adversaries, and has no such robustness against nonce misuse. We think specifying both NAE and MRAE sharing the principle components (TBC and its state update function, padding function etc.) is useful in practice as it allows users to choose and switch between them depending on their needs.
- **Small messages.** The variants of TGIF are efficient for small-message scenario. For example: TGIF-N1 and TGIF-N2 require 3 and 4 TBC calls, respectively, for processing 1 block of AD and 1 block of message.

- **Simplicity/Small footprint.** TGIF has a quite small footprint. Especially for TGIF-N, we essentially need what is needed to implement the TBC TGIF-TBC itself, and one-block mask exclusively used by TGIF-N2 and TGIF-M2. We remark that this becomes possible thanks to the permutation-based structure of TGIF-TBC's tweakey schedule, which allows to share the state registers used for storing input variable and for deriving round-key values. Thus, this feature is specific to our use of TGIF-TBC, though one can expect a similar effect with TBC using a simple tweak(ey) schedule. We do not need the inverse circuit for TGIF-TBC which was needed for ΘCB3. A comparison in Section 6 (Table 6.1) shows that TGIF-N is quite small and especially efficient in terms of a combined metric of size and speed, compared with other schemes.

  TGIF-M also has a small footprint due to the shared structure with TGIF-N.

- **Flexibility.** TGIF has a large flexibility. Generally, it is defined as a generic mode for TBCs, and the security proofs under ICM contribute to a high confidence of the scheme when combined with a secure TBC under related-tweakey model. In fact, all mode versions of TGIF (and some others) are used in the Remus design [12] with a different underlying TBC.

- **Side channels.** TGIF does not inherently guarantee security against Side Channel Analysis and Fault Attacks. However, standard countermeasures are easily adaptable for TGIF, e.g. Fresh Rekeying [19], Masking [21], etc. Moreover, powerful fault attacks that require a small number of faults and pairs of faulty and non-faulty ciphertexts, such as DFA, are not applicable to TGIF-N without violating the security model, i.e. misusing the nonce or releasing unverified plaintexts. We plan to study the applicability/cost of such countermeasures, in addition to newly proposed countermeasures suited specifically for TGIF in subsequent works.

# Design Rationale

TGIF is designed with the following goals in mind:

1. Have a small-footprint and efficient TBC internal primitive, while providing strong security guarantees against state-of-the-art attacks.
2. Have a very small area compared to other TBC/BC based AEAD modes, while providing security proofs in the well-established ideal-cipher model.

## 6.1 TGIF-TBC Design

Our cipher TGIF-TBC can be seen as an improvement over GIFT, while enabling tweakable capability. This work was initiated by the new observation of a special structure in GIFT cipher, that can greatly improve its performance in software. Namely, instead of performing the normal bit permutation of GIFT (efficient on hardware, but can be slow on software), one can instead use an alternative representation during 4 rounds, that will eventually have a completely equivalent effect as the normal 4 rounds of GIFT. This new representation, bitslice in essence, only uses very simple rotations which are easy to perform in software.

In other words, we retain all the benefits from the structure and the analysis of GIFT (ability to easily get a number of active Sboxes for differential and linear cryptanalysis, etc.), while having a very efficient alternative bitslice representation of it. TGIF-TBC can be seen as a link between SPN ciphers and AND-Rotation-XOR ciphers.

We note that this type of special bitslice representation of GIFT is simple for its 64-bit version, but not for its 128-bit one. This is the reason why we chose to use GIFT-64 as basic building block (and a Misty structure to propose a 128-bit cipher). Besides, this representation goes back to identity after 4 rounds of GIFT, which is why we used 4 rounds inside the GIFT-64 black-box. Overall, our cipher is notably more efficient than GIFT-128, one of the very lightest cipher recently published, while actually ensuring a higher number of active Sboxes per Sbox computed. The analysis is also simpler for related-key attacks, which allows use to enable the crucial tweak capability, that leads to very efficient and lightweight authenticated encryption modes, such as REMUS.

The tweakey schedule of TGIF-TBC has been chosen so as to minimize the number of XORs, while proposing acceptable diffusion in the tweakey state. It is also very simple and efficient to compute this tweakey schedule on the fly. Besides, we forced the special feature that the tweakey schedule naturally makes the tweakey state to come back to its initial value at the end of the cipher computation. This has two benefits in practice. First, for decryption the tweakey passed as input is the tweakey state to start the decryption from (no need to compute the entire tweakey schedule for all rounds in order to obtain the stating decryption tweakey state). Secondly, in serial modes like REMUS, saving memory is crucial for lightweight performances. Since TGIF-TBC will receive some special counter-like value to its tweakey input, this key schedule feature will allow to avoid

allocating extra memory to compute the cipher: the counter register can directly be used for free as tweakey state.

## 6.2 Mode Design

**Rationale of NAE Mode.** By seeing TGIF-N as a mode of TBC (ICE), TGIF-N has a similar structure as a mode called iCOFB, which appeared in the full version of CHES 2017 paper [8]. Because it was introduced to show the feasibility of the main proposal of [6], block cipher mode COFB, it does not work as a full-fledged AE using conventional TBCs. Therefore, starting from iCOFB, we apply numerous changes for improving efficiency while achieving high security. As a result, TGIF-N becomes a much more advanced, sophisticated NAE mode based on a TBC. Assuming ICE is an ideally secure TBC, the security bound of TGIF-N is essentially equivalent to ΘCB3, having full $n$-bit security. The remaining problem is how to efficiently instantiate TBC. We could use a dedicated TBC, or a conventional block cipher mode (such as XEX [24]), however, they have certain limitations on security and efficiency. To overcome such limitations, we choose to use a block cipher with tweak-dependent key/mask derivation. This approach, initiated by Mennink [20], enables the performance that cannot be achieved by the previous approaches, at the expense of the ideal-cipher model for security. Specifically, the TBC ICE has 2 variants, ICE1 and ICE2, that can be seen as a variant of XHX [14]. Each variant has its own security level, namely, ICE1 has $n/2$-bit security and ICE2 has $n$-bit security. They have different computation cost for key/mask derivations and have different state sizes. Given the $n$-bit security of *outer* TBC-based mode, the standard hybrid argument shows that the security of TGIF-N is effectively determined by the security of the internal ICE.

**Rationale of MRAE Mode.** TGIF-M is designed as an MRAE mode following the structure of SIV [26] and SCT [22]. TGIF-M reuses the components of TGIF-N as much as possible to inherit its implementation advantages and the security. In fact, this brings us several advantages (not only for implementation aspects) over SIV/SCT. TGIF-M needs an equivalent number of primitive calls as SCT. The difference is in the primitive: TGIF-M uses an $n$-bit block cipher while SCT uses an $n$-bit-block dedicated TBC. Moreover, TGIF-M has a smaller state than SCT because of single-state encryption part taken from TGIF-N (SCT employs a variant of counter mode). Similarly to TGIF-N, the provable security of TGIF-M is effectively determined by the internal ICE. For TGIF-M2, thanks to $n$-bit security of ICE2, its security is equivalent to SCT: the security depends on the maximum number of repetition of a nonce in encryption ($r$), and if $r = 1$ (*i.e.*, NR adversary), we have the full $n$-bit security. Security will gradually decrease as $r$ increases, also known as "graceful degradation", and even if $r$ equals to the number of encryption queries, implying nonces are fixed, we maintain the birthday-bound, $n/2$-bit security. For TGIF-M1, the security is $n/2$ bits for both NR and NM adversaries due to the $n/2$-bit security of ICE1.

ZAE [13] is another TBC-based MRAE. Although it is faster than SCT, the state size is much larger than SCT and TGIF-M.

**Efficiency Comparison.** In Table 6.1, we compare TGIF-N to ΘCB3, COFB, Beetle and Ascon, where state size is the minimum number of bits that the mode has to maintain during its operation, and rate is the number of bits generated by the underlying permutation for every input block, *i.e.*, it is inversely proportional to the permutation size. ΘCB3 is a well-studied TBC-based AEAD mode. COFB is a BC-based lightweight AEAD mode. Beetle is a Sponge-based AEAD mode, but it holds a lot of resemblance to TGIF-N. The comparison follows the following guidelines, while trying to be fair in comparing designs that follow completely different approaches:

1. $k = 128$ for all the designs.
2. $n$ is the input block size (in bits) for each primitive call.

**Table 6.1:** Features of TGIF-N members compared to other TBC-based AEAD algorithms: $\lambda$ is the bit security level of a mode.

| Scheme | Number of Primitive Calls | State Size (S) | Rate (R) | R/S | Inverse Free? |
|---|---|---|---|---|---|
| TGIF-N1 | $\lceil \frac{\lvert A\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$ | $2n = 4\lambda$ | $\frac{1}{n} = \frac{1}{2\lambda}$ | $\frac{1}{8\lambda^2}$ | Yes |
| TGIF-N2 | $\lceil \frac{\lvert A\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 2$ | $3n = 3\lambda$ | $\frac{1}{n} = \frac{1}{\lambda}$ | $\frac{1}{3\lambda^2}$ | Yes |
| COFB [7] | $\lceil \frac{\lvert A\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$ | $2.5n = 5\lambda$ | $\frac{1}{n} = \frac{1}{2\lambda}$ | $\frac{1}{10\lambda^2}$ | Yes |
| ΘCB3 [1][17] | $\lceil \frac{\lvert A\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$ | $4.5n = 4.5\lambda$ | $\frac{1}{n} = \frac{1}{\lambda}$ | $\frac{1}{4.5\lambda^2}$ | No |
| Beetle [2][5] | $\lceil \frac{\lvert A\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 2$ | $2n = 2.12\lambda$ | $\frac{1}{2n} = \frac{1}{2.12\lambda}$ | $\frac{1}{4.47\lambda^2}$ | Yes |
| Ascon-128 [9] | $\lceil \frac{\lvert A\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$ | $7n = 3.5\lambda$ | $\frac{1}{5n} = \frac{1}{2.5\lambda}$ | $\frac{1}{8.75\lambda^2}$ | Yes |
| Ascon-128a [9] | $\lceil \frac{\lvert A\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$ | $3.5n = 3.5\lambda$ | $\frac{1}{2.5n} = \frac{1}{2.5\lambda}$ | $\frac{1}{8.75\lambda^2}$ | Yes |

[1] $t = 128, d = 64$; [2] $n = 128$

**Table 6.2:** Features of TGIF-M members compared to SCT/ZAE: $\lambda$ is the bit security level of a mode.

| Scheme | Number of Primitive Calls | State Size (S) | Rate (R) | R/S | Inverse Free? |
|---|---|---|---|---|---|
| TGIF-M1 | $\lceil \frac{\lvert A\rvert+\lvert M\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$ | $2n = 4\lambda$ | $\frac{1}{n} = \frac{1}{2\lambda}$ | $\frac{1}{8\lambda^2}$ | Yes |
| TGIF-M2 | $\lceil \frac{\lvert A\rvert+\lvert M\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 2$ | $3n = 3\lambda$ | $\frac{1}{n} = \frac{1}{\lambda}$ | $\frac{1}{3\lambda^2}$ | Yes |
| SCT [1][23] | $\lceil \frac{\lvert A\rvert+\lvert M\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$ | $4n = 4\lambda$ | $\frac{1}{n} = \frac{1}{\lambda}$ | $\frac{1}{4\lambda^2}$ | Yes |
| ZAE [2][13] | $\lceil \frac{\lvert A\rvert+\lvert M\rvert}{2n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 6$ | $7n = 7\lambda$ | $\frac{1}{n} = \frac{1}{\lambda}$ | $\frac{1}{7\lambda^2}$ | Yes |

[1] $t = 128, \tau = 128$; [2] $t = 128$

3. $\lambda$ is the security level of the design.
4. For BC/TBC based designs, the key is considered to be stored inside the design, but we also consider that the encryption and decryption keys are interchangeable, *i.e.*, the encryption key can be derived from the decryption key and vice versa. Hence, there is no need to store the master key in additional storage. The same applies for the nonce.
5. For Sponge and Sponge-like designs, if the key/nonce are used only during initialization, then they are counted as part of the state and do not need extra storage. However, in designs like Ascon, where the key is used again during finalization, we assume the key storage is part of the state, as the key should be supplied only once as an input.

Our comparative analysis of these modes show that TGIF-N achieves its goals, as TGIF-N1 has $2n$ state, which is smaller than COFB and equal to Beetle. TGIF-N1 and COFB both have birthday security, *i.e.*, $n/2$. Beetle achieves higher security, at the expense of using a $2n$-bit permutation. Our analysis also shows that among the considered AEAD modes, TGIF-N2 achieves the highest R/S ratio, with a state size of $3n$ but only an $n$-bit permutation.

Similar comparison is shown in Table 6.2 for Misuse-Resistant TBC-based AEAD modes. It shows that TGIF-M2 particularly is very efficient.

### 6.2.1 Hardware Implementations

The goal of the design of TGIF is to have a very small area overhead over the underlying TBC, specially for the round-based implementations. In order to achieve this goal, we set two requirements:

1. There should be no extra Flip-Flops over what is already required by the TBC, since Flip-Flops

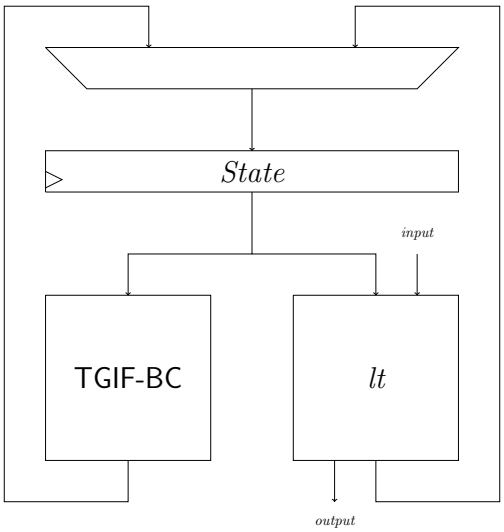


**Figure 6.1:** Expected architectures for TGIF-TBC and TGIF.

are very costly ($4 \sim 7$ GEs per Flip-Flop).

2. The number of possible inputs to each Flip-Flop and outputs of the circuits have to be minimized. This is in order to reduce the number of multiplexers required, which is usually one of the cause of efficiency reduction between the specification and implementation.

In this section, we describe various design choices that help achieve these two goals.

**General Architecture and Hardware Estimates.** The mode was designed with the architecture in Figure 6.1 in mind, where only a full-width state-register is used, carrying the TBC state and tweakey values, and every cycle, it is either kept without change, updated with the TBC round output (which includes a single round of the key scheduling algorithm) or the output of a simple linear transformation, which consists of $\rho/\rho^{-1}$ and the block counter.

**Hardware Cost of TGIF-N1.** The overhead of TGIF-N1 is mostly due to the doubling (3 XORs) and $\rho$ operations (68 XORs, assuming the input/output bus has width of 32 bits). Moreover, we need 2 128-bit multiplexers to select the input to the tweakey out of four positive values: $K$, $S$ (after applying the KDF-ICE function), $lt$, or the TGIF-TBC round key. We assume a multiplexer costs $\sim 2.75$ GEs and an XOR gate costs $\sim 2.25$ GEs. In total, this adds up to $\sim 864$ GEs on top of TGIF-TBC.

In order to design a combined encryption/decryption circuit, we show below that the decryption costs only extra 32 multiplexers and $\sim 32$ OR gates, or $\sim 100$ GEs.

**Hardware Cost of TGIF-N2.** TGIF-N2 is similar to TGIF-N1, with an additional mask $V$. Hence, the additional cost comes from the need to store and process this mask. The storage cost is simply 128 extra Flip-Flops. However, the processing cost can be tricky, especially since we adopt a serial concept for the implementation of $\rho$. Hence, we also adopt a serial concept for the processing of $V$. We assume that $V$ will be updated in parallel to $\rho$ and we merge the masking and $\rho$ operations. Consequently, we need 64 XORs for the masking, 3 XORs for doubling, 5 XORs in order to correct the domain separation bits after each block (note that 3 bits are fixed), and 1 Flip-Flop for the serialization of doubling. Overall, we need $\sim 800$ GEs on top of TGIF-N1, which is again smaller than almost all other AEAD designs (except other TGIF variants), while achieving BBB security.

**Algorithm** KDF-ICE2$(N, K)$

1. $S \leftarrow 0^n$
2. $(S, \eta) \leftarrow \rho(S, N)$
3. $S \leftarrow E_K(S)$
4. $(S, L) \leftarrow \rho(S, 0^n)$
5. $S \leftarrow E_{K \oplus 1}(S)$
6. $(S, V) \leftarrow \rho(S, 0^n)$
7. $S \leftarrow 0^n$
8. **return** $(L, V)$

**Figure 6.2:** Alternative description of KDF-ICE2. KDF-ICE1 is obtained by removing lines 5 and 6.

### 6.2.2 Primitives Choices

**LFSR-Based Counters.** The NIST call for lightweight AEAD algorithms requires that such algorithms must allow encrypting messages of length at least $2^{50}$ bytes while still maintaining their security claims. This means that using TBCs whose block sizes are 128 and 64 bits, we need a block counter of a period of at least $2^{46}$ and $2^{47}$, respectively. While this can be achieved by a simple arithmetic counter of 46 bits, arithmetic counters can be costly both in terms of area ($3 \sim 5$ GEs/bit) and performance (due to the long carry chains which limit the frequency of the circuit). In order to avoid this, we decided to use LFSR-based counters, which can be implemented using a handful of XOR gates (3 XORs $\approx 6 \sim 9$ GEs). These counters are consecutive doubling of the key $L$, which is equivalent to a Galois LFSR. This, in addition to the architecture described above, makes the cost of counter almost negligible.

**Tag Generation.** While TGIF has a lot of similarities compared to iCOFB, the original iCOFB simply outputs the final chaining value as the tag. Considering hardware simplicity, we changed it so that the tag is the final output state (*i.e.*, the same way as the ciphertext blocks). In order to avoid branching when it comes to the output of the circuit, the tag is generated as $G(S)$ instead of $S$. In hardware, this can be implemented as $\rho(S, 0^n)$, *i.e.*, similar to the encryption of a zero vector. Consequently, the output bus is always connected to the output of $\rho$ and a multiplexer is avoided.

**Mask Generation in** KDF-ICE**.** Similar to tag generation, we generate the masks by applying $G$ to a standard $2n$-bit keyed permutation. The reason for that is to be able to reuse the same circuit used for the normal opration of TGIF for KDF-ICE. Moreover, it allows us to easily output and store the masks during the first pass of TGIF-N, to be used during the second pass. Effectively, KDF-ICE2 is equivalent to the algorithm in Figure 6.2, where for KDF-ICE1 lines 5 and 6 will be removed. This algorithm shows that the KDF has the same structure as the main encryption/decryption part of TGIF itself and the same hardware circuit can be very easily reused with almost no overhead.

**Padding.** The padding function used in TGIF is chosen so that the padding information is always inserted in the most significant byte of the last block of the message/AD. Hence, it reduces the number of decisions for each byte to only two decisions (either the input byte or a zero byte, except the most significant byte which is either the input byte or the byte length of that block). Besides, it is also the case when the input is treated as a string of words (16-, 32-, 64- or 128-bit words). This is much simpler than the classical $10^*$ padding approach, where every word has a lot of different possibilities when it comes to the location of the padding string. Besides, usually implementations maintain the length of the message in a local variable/register, which means that the padding information is already available, just a matter of placing it in the right place in the message, as opposed to the decoder required to convert the message length into $10^*$ padding.

**Padding Circuit for Decryption.** One of the main features of TGIF is that it is inverse free and both the encryption and decryption algorithms are almost the same. However, it can be tricky to understand the behavior of decryption when the last ciphertext block has length $< n$. In order to understand padding in decryption, we look at the $\rho$ and $\rho^{-1}$ functions when the input plaintext/ciphertext is partial. The $\rho$ function applied on a partial plaintext block is shown in Equation (6.1). If $\rho^{-1}$ is directly applied to $\texttt{pad}_n(C)$, the corresponding output will be incorrect, due to the truncation of the last ciphertext block. Hence, before applying $\rho^{-1}$ we need to regenerate the truncated bits. It can be verified that $C^{'} = \texttt{pad}_n(C) \oplus \texttt{msb}_{n-|C|}(G(S))$. Once $C^{'}$ is regenerated, $\rho^{-1}$ can be computed as shown in Equation (6.3)

$$\begin{bmatrix} 1 & 1 \\ 1 & G \end{bmatrix} \begin{bmatrix} S \\ \texttt{pad}_n(M) \end{bmatrix} = \begin{bmatrix} S' \\ C' \end{bmatrix}, \tag{6.1}$$

$$C = \texttt{lsb}_{|M|}(C^{'}). \tag{6.2}$$

$$C^{'} = \texttt{pad}_n(C) \oplus \texttt{msb}_{n-|C|}(G(S)), \tag{6.3}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \oplus G \end{bmatrix} \begin{bmatrix} S \\ C' \end{bmatrix} = \begin{bmatrix} S' \\ M \end{bmatrix}. \tag{6.4}$$

While this looks like a special padding function, in practice it is simple. First of all, $G(S)$ needs to be calculated anyway. Besides, the whole operation can be implemented in two steps:

$$M = C \oplus \texttt{lsb}_{|C|}(G(s)), \tag{6.5}$$

$$S^{'} = \texttt{pad}_n(M) \oplus S, \tag{6.6}$$

which can have a very simple hardware implementation, as discussed in the next paragraph.

**Encryption-Decryption Combined Circuit.** One of the goals of TGIF is to be efficient for implementations that require a combine encryption-decryption datapath. Hence, we made sure that the algorithm is inverse free, *i.e.*, it does not use the inverse function of TGIF-TBC or $G(S)$. Moreover, $\rho$ and $\rho^{-1}$ can be combined using only one multiplexer, whose size depends on the size of the input/output bus. The same circuit can be used to solve the padding issue in decryption, by padding $M$ instead of $C$. The tag verification operation simply checks that if $\rho(S, 0^n)$ equals to $T$, which can be serialized depending on the implementation of $\rho$.

**Choice of the $G$ Matrix.** We chose the position of $G$ so that it is applied to the output state. This removes the need of $G$ for AD processing, which improves (e.g.) software performance. In Section 4, we listed the security condition for $G$, and we choose our matrix $G$ so that it meets these conditions and suits well for various hardware and software.

We noticed that for lightweight applications, most implementations use an input/output bus of width $\leq 32$. Hence, we expect the implementation of $\rho$ to be serialized depending on the bus size. Consequently, the matrix used in iCOFB can be inefficient as it needs a feedback operation over 4 bytes, which requires up to 32 extra Flip-Flops in order to be serialized, something we are trying to avoid in TGIF. Moreover, the serial operation of $\rho$ is different for byte, which requires additional multiplexers.

However, we observed that if the input block is interpreted in a different order, both problems can be avoided. First, it is impossible to satisfy the security requirements of $G$ without any feedback signals, *i.e.*, $G$ is a bit permutation.

- If $G$ is a bit permutation with at least one bit going to itself, then there is at least one non-zero value on the diagonal, so $I + G$ has at least 1 row that is all 0s.
- If $G$ is a bit permutation without any bit going to itself, then every column in $I + G$ has exactly two 1's. The sum of all rows in such matrix is the 0 vector, which means the rows are linearly dependent. Hence, $I + G$ is not invertible.

However, the number of feedback signals can be adjusted to our requirements, starting from only 1 feedback signal. Second, we noticed that the input block/state of length $n$ bits can be treated as several independent sub-blocks of size $n/w$ each. Hence, it is enough to design a matrix $G_s$ of size $w \times w$ bits and apply it independently $n/w$ times to each sub-block. The operation applied on each sub-block in this case is the same, (*i.e.*, as we can distribute the feedback bits evenly across the input block). Unfortunately, the choice of $w$ and $G_s$ that provides the optimal results depends on the implementation architecture. However, we found out that the best trade-off/balance across different architectures is when $w = 8$ and $G_s$ uses a single bit feedback.

In order to verify our observations, we generated a family of matrices with different values of $w$ and $G_s$, and measured the cost of implementing each of them on different architectures.

# Implementations (To be added soon)

## 7.1 Software Performances

### 7.1.1 Software implementations

To be added soon.

### 7.1.2 Micro-controller implementations

To be added soon.

## 7.2 Hardware Performances

### 7.2.1 ASIC implementations

To be added soon.

### 7.2.2 FPGA implementations

To be added soon.

# Acknowledgments

The second and fourth authors are supported by the Temasek Labs grant (DSOCL16194).

# Bibliography


[1] Banik, S., Pandey, S.K., Peyrin, T., Sasaki, Y., Sim, S.M., Todo, Y.: Gift: a small present. In: International Conference on Cryptographic Hardware and Embedded Systems, Springer (2017) 321–345

[2] Banik, S., Pandey, S.K., Peyrin, T., Sasaki, Y., Sim, S.M., Todo, Y.: GIFT: A small present - towards reaching the limit of lightweight encryption. In: CHES. Volume 10529 of Lecture Notes in Computer Science., Springer (2017) 321–345

[3] Bellare, M., Namprempre, C.: Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. J. Cryptology **21**(4) (2008) 469–491

[4] Bellare, M., Rogaway, P., Wagner, D.A.: The EAX mode of operation. In: FSE. Volume 3017 of Lecture Notes in Computer Science., Springer (2004) 389–407

[5] Chakraborti, A., Datta, N., Nandi, M., Yasuda, K.: Beetle family of lightweight and secure authenticated encryption ciphers. IACR Transactions on Cryptographic Hardware and Embedded Systems (2018) 218–241

[6] Chakraborti, A., Iwata, T., Minematsu, K., Nandi, M.: Blockcipher-based authenticated encryption: How small can we go? In: CHES. Volume 10529 of Lecture Notes in Computer Science., Springer (2017) 277–298

[7] Chakraborti, A., Iwata, T., Minematsu, K., Nandi, M.: Blockcipher-based authenticated encryption: How small can we go? In: International Conference on Cryptographic Hardware and Embedded Systems, Springer (2017) 277–298

[8] Chakraborti, A., Iwata, T., Minematsu, K., Nandi, M.: Blockcipher-based authenticated encryption: How small can we go? (full version of [6]). IACR Cryptology ePrint Archive **2017** (2017) 649

[9] Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1. 2. Submission to the CAESAR Competition (2016)

[10] George Mason University: ATHENa: Automated Tools for Hardware EvaluatioN. https://cryptography.gmu.edu/athena/ (2017)

[11] Handschuh, H., Preneel, B.: Key-Recovery Attacks on Universal Hash Function Based MAC Algorithms. In Wagner, D., ed.: Advances in Cryptology - CRYPTO 2008. Volume 5157., Springer (2008) 144–161

[12] Iwata, T., Khairallah, M., Minematsu, K., Peyrin, T.: Remus. A submission to NIST Lightweight Cryptography (2019)

[13] Iwata, T., Minematsu, K., Peyrin, T., Seurin, Y.: Zmac: a fast tweakable block cipher mode for highly secure message authentication. In: Annual International Cryptology Conference, Springer (2017) 34–65

[14] Jha, A., List, E., Minematsu, K., Mishra, S., Nandi, M.: XHX - A framework for optimally secure tweakable block ciphers from classical block ciphers and universal hashing. **(to appear)**

[15] Joux, A.: Authentication Failures in NIST Version of GCM. Comments submitted to NIST Modes of Operation Process (2006) Available at http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/800-38_Series-Drafts/GCM/Joux_comments.pdf.

[16] Khairallah, M., Chattopadhyay, A., Peyrin, T.: Looting the luts: Fpga optimization of aes and aes-like ciphers for authenticated encryption. In: International Conference in Cryptology in India, Springer (2017) 282–301

[17] Krovetz, T., Rogaway, P.: The software performance of authenticated-encryption modes. In: International Workshop on Fast Software Encryption, Springer (2011) 306–327

[18] Kumar, S., Haj-Yihia, J., Khairallah, M., Chattopadhyay, A.: A comprehensive performance analysis of hardware implementations of caesar candidates. IACR Cryptology ePrint Archive **2017** (2017) 1261

[19] Medwed, M., Standaert, F.X., Großschädl, J., Regazzoni, F.: Fresh re-keying: Security against side-channel and fault attacks for low-cost devices. In: International Conference on Cryptology in Africa, Springer (2010) 279–296

[20] Mennink, B.: Optimally secure tweakable blockciphers. In: FSE. Volume 9054 of Lecture Notes in Computer Science., Springer (2015) 428–448

[21] Messerges, T.S.: Securing the aes finalists against power analysis attacks. In: International Workshop on Fast Software Encryption, Springer (2000) 150–164

[22] Peyrin, T., Seurin, Y.: Counter-in-tweak: Authenticated encryption modes for tweakable block ciphers. In: CRYPTO (1). Volume 9814 of Lecture Notes in Computer Science., Springer (2016) 33–63

[23] Peyrin, T., Seurin, Y.: Counter-in-tweak: authenticated encryption modes for tweakable block ciphers. In: Annual International Cryptology Conference, Springer (2016) 33–63

[24] Rogaway, P.: Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In: ASIACRYPT. Volume 3329 of Lecture Notes in Computer Science., Springer (2004) 16–31

[25] Rogaway, P.: Nonce-based symmetric encryption. In: FSE. Volume 3017 of Lecture Notes in Computer Science., Springer (2004) 348–359

[26] Rogaway, P., Shrimpton, T.: A provable-security treatment of the key-wrap problem. In: EUROCRYPT. Volume 4004 of Lecture Notes in Computer Science., Springer (2006) 373–390

[27] Wang, L., Guo, J., Zhang, G., Zhao, J., Gu, D.: How to build fully secure tweakable blockciphers from classical blockciphers. In: ASIACRYPT (1). Volume 10031 of Lecture Notes in Computer Science. (2016) 455–483

# Appendix

## 7.3 Domain Separation

**Table 7.1:** Domain separation byte $B$ of TGIF. Bit $b_6$ is to be set to the appropriate value according to the parameter sets.

| | $b_7$ | $b_6$ | $b_5$ | $b_4$ | $b_3$ | $b_2$ | $b_1$ | $b_0$ | int($B$) | case |
|---|---|---|---|---|---|---|---|---|---|---|
| TGIF-N | 0 | - | 0 | 0 | 0 | 1 | 0 | 0 | 4 | A main |
| | 0 | - | 0 | 0 | 1 | 1 | 0 | 0 | 12 | A last unpadded |
| | 0 | - | 0 | 0 | 1 | 1 | 0 | 1 | 13 | A last padded |
| | 0 | - | 0 | 0 | 0 | 0 | 1 | 0 | 2 | M main |
| | 0 | - | 0 | 0 | 1 | 0 | 1 | 0 | 10 | M last unpadded |
| | 0 | - | 0 | 0 | 1 | 0 | 1 | 1 | 11 | M last padded |
| TGIF-M | 0 | - | 1 | 0 | 0 | 1 | 0 | 0 | 36 | A main |
| | 0 | - | 1 | 0 | 1 | 1 | 0 | 0 | 44 | A last unpadded |
| | 0 | - | 1 | 0 | 1 | 1 | 0 | 1 | 45 | A last padded |
| | 0 | - | 1 | 0 | 0 | 1 | 1 | 0 | 38 | M auth main |
| | 0 | - | 1 | 0 | 1 | 1 | 1 | 0 | 46 | M auth last unpadded |
| | 0 | - | 1 | 0 | 1 | 1 | 1 | 1 | 47 | M auth last padded |
| | 0 | - | 1 | 0 | 0 | 0 | 1 | 0 | 34 | M enc main |

# Changelog

- 29-03-2019: version v1.0