SUNDAE-GIFT

v1.0

Subhadeep Banik[3], Andrey Bogdanov[4,5], Thomas Peyrin[1,2], Yu Sasaki[6], Siang Meng Sim[1], Elmar Tischhauser[5], and Yosuke Todo[6]

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

[2] Temasek Labs@NTU, Singapore

[3] LASEC, École Polytechnique Fédérale de Lausanne, Switzerland
subhadeep.banik@epfl.ch

[4] Technical University of Denmark, Denmark
anbog@dtu.dk

[5] Cybercrypt A/S, Denmark
elmar@cyber-crypt.com

[6] NTT Secure Platform Laboratories, Japan
Sasaki.Yu@lab.ntt.co.jp
yosuke.todo.xt@hco.ntt.co.jp

Table of Contents

1 Introduction

Lightweight block cipher design is one of the most mature research areas, with constructions going back to 2007, optimizing for a variety of efficiency goals such as latency [11], area [15], and energy [3]. Although optimizing block cipher design is an important first step, block ciphers on their own are only building blocks, and should be used in modes of operation to achieve security. In particular, ensuring data confidentiality and authenticity is done using an *authenticated encryption* (AE) mode of operation.

Even if a block cipher is ideally suited to a given environment when considered in isolation, it could be used in an AE mode of operation which erases many of the block cipher's benefits. In fact, AE modes are often not designed to account for the different requirements imposed by lightweight settings. The mode might require two separate, independent keys, as with SIV [30], a state size of at least thrice that of the underlying block size, as with COPA [2], or multiple initial block cipher calls before it can start processing data, like in EAX [12].

Exceptions include the AE modes CLOC [21], JAMBU [33], and COFB [16], which try to reduce state size and number of block cipher calls to optimize for short messages. However, the challenges imposed by constrained environments are not limited to efficiency constraints, as fundamental security assumptions might be difficult to guarantee as well. For example, devices might lack proper randomness sources, or have limited secure storage to maintain state, in which case they might not be able to generate the nonces necessary to ensure that modes such as CLOC, JAMBU, and COFB maintain security. In such cases, algorithms which provide more robust security are better, such as nonce-misuse resistant AE [30], as they do not fail outright in the wrong conditions.

An efficient nonce-misuse resistant dedicated instantiation of SIV called GCM-SIV was proposed by Gueron and Lindell at CCS 2015 [19]. While it attains very competitive performance in software on recent Intel architectures, it requires full multiplications in $GF(2^{128})$, which makes the scheme unattractive in hardware and on resource-constrained platforms. The importance of good implementation characteristics on all platforms was already pointed out in [25]: the same cryptographic algorithms used on the small devices of the Internet of Things also have to be employed on the servers that are communicating with them. Crucially, however, the few designs explicitly aiming at being simultaneously efficient on lightweight as well as high-performance platforms such as [14,24] do not provide nonce-misuse resistant authenticated encryption.

We introduce an AE mode of operation, `SUNDAE-GIFT`, which

1. competes with CLOC and JAMBU in number of block cipher calls for short messages,
2. improves over those algorithms and COFB in terms of state size,
3. provides maximal robustness to a lack of proper randomness or secure state, and
4. simultaneously offers good implementation characteristics on lightweight and high-performance platforms.

`SUNDAE-GIFT` is based on the mode of operation `SUNDAE` [4], introduced in ToSC 2019. We choose the block cipher `GIFT-128` [9] as the underlying block cipher because of the natural advantages it offers on lightweight platforms. `GIFT` improves over `PRESENT` in both security and efficiency. Interestingly, `GIFT` offers extremely good performances and even surpasses both `SKINNY` and `SIMON` for round-based implementations (see Table 1). This indicates that `GIFT` is probably the cipher the most suited for the very important low-energy consumption use cases. Due to its simplicity and natural bitslice organisation of the inner data flow, our cipher is very versatile and performs also very well on software, reaching similar performances as `SIMON` [9], the current fastest lightweight candidate on software.

Table 1: Hardware performances of round-based implementations of PRESENT, SKINNY, SIMON and our new cipher GIFT, synthesized with STM 90nm Standard cell library.

	Area (GE)	Delay (ns)	Cycles	TP_{MAX} (MBit/s)	Power (μW) (@10MHz)	Energy (pJ)
GIFT-64-128	1345	1.83	29	1249.0	74.8	216.9
SKINNY-64-128	1477	1.84	37	966.2	80.3	297.0
PRESENT 64/128	1560	1.63	33	1227.0	71.1	234.6
SIMON 64/128	1458	1.83	45	794.8	72.7	327.3
GIFT-128-128	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

For the rest of this document, we refer the SUNDAE publication in ToSC2018 [4] as the SUNDAE paper and the GIFT article posted on Cryptology ePrint Archive [10] (the full version of the publication in CHES2017 [9]) as the GIFT paper.

2 Specification

2.1 Notations

Unless specified otherwise, all sets are finite. If X is a set, then X^n is the set of length-n sequences of X, $\mathsf{X}^{\leq q}$ the set of sequences of X of length not greater than q including the empty sequence, and X^* the set of finite-length sequences of X. If $X \in \mathsf{X}^*$, then $|X|$ is its length. Given $X, Y \in \mathsf{X}^*$, concatenation of X and Y is denoted $X\|Y$, or simply XY when no confusion arises.

The notation $x \mapsto y$ is used to denote a function which maps the symbol x on the left to the symbol y on the right. If f is a function with domain $\mathsf{X} \times \mathsf{Y}$, then we write $f(X, Y)$ and $f_X(Y)$ interchangeably, and use the notation f_X to denote the function obtained by fixing the first input of f to X.

Throughout the paper, $n = 128$ denotes *block size*. The set of *blocks* is $\{0,1\}^{\leq n}$, and $\mathsf{B} := \{0,1\}^n$ denotes the subset of *complete* blocks, with all other blocks called *incomplete*. The element $0^n \in \mathsf{B}$ denotes the complete block consisting of only zeroes.

The empty string is denoted ε. Given two equal-length elements $X, Y \in \{0,1\}^*$, $X \oplus Y$ denotes their bitwise XOR. If $X \in \{0,1\}^*$, then $\lfloor X \rfloor_m$ denotes truncating X to the m most significant bits of X. Splitting a non-empty string X into blocks is done by computing its *block length* ℓ, which is the smallest integer greater than or equal to $|X|/n$, and processing X as

$$X[1]X[2]\cdots X[\ell-1]X[\ell] \xleftarrow{n} X$$

where $|X[i]| = n$ for $1 \leq i < \ell$, and $0 < |X[\ell]| \leq n$.

A 128-bit block $X \in \mathsf{B}$ can further be expressed as 4 32-bit segments, 8 16-bit words, 16 bytes or 128 bits as follows

$$\begin{aligned} S_0S_1S_2S_3 &\xleftarrow{32} X \\ W_0W_1\cdots W_6W_7 &\xleftarrow{16} X \\ B_0B_1\cdots B_{14}B_{15} &\xleftarrow{8} X \\ b_{127}b_{126}\cdots b_1b_0 &\xleftarrow{1} X, \end{aligned}$$

where b_{127} being the most significant bit of $X[i]$. Note that only the indexing of the bits is in the reverse order.

Padding. The function $\mathsf{pad} : \{0,1\}^{\leq n} \to \mathsf{B}$ pads an incomplete block X with a 1 followed by $127 - |X|$ zeroes, and leaves complete blocks as-is:

$$\mathsf{pad}(X) = \begin{cases} X \parallel 10^{127-|X|} & \text{if } |X| < n \\ X & \text{otherwise}\,. \end{cases}$$

Multiplier. Given $X \in \mathsf{B}$, we let $2 \times X$ and $4 \times X$ denote multiplications defined as the following

$$2 \times (B_0 \| B_1 \| \cdots \| B_{15}) = B_1 \| B_2 \| \cdots \| B_{10} \| B_{11} \oplus B_0 \| B_{12} \| B_{13} \oplus B_0 \| B_{14} \| B_{15} \oplus B_0 \| B_0,$$

and $4 \times X = 2 \times (2 \times X)$.

Block Cipher Encryption. The function $\mathsf{E} : \mathsf{K} \times \mathsf{B} \to \mathsf{B}$ denotes `GIFT-128` encryption, with $\mathsf{K} := \{0,1\}^{128}$ the set of keys.

Conditional. The expression $a \ ? \ b : c$ evaluates to b if a is true and c otherwise.

2.2 The AEAD Scheme `SUNDAE-GIFT`

As the name suggested, `SUNDAE-GIFT` is a family of AEAD schemes that is based on the AEAD scheme `SUNDAE` with `GIFT-128` as the underlying block cipher.

Parameter Sets. There are four members in the `SUNDAE-GIFT` family, as seen in the following

Member	Name	Nonce length	Key length	Tag length
1 (primary)	`SUNDAE-GIFT-96`	96	128	128
2	`SUNDAE-GIFT-0`	0	128	128
3	`SUNDAE-GIFT-128`	128	128	128
4	`SUNDAE-GIFT-64`	64	128	128

Notice that `SUNDAE-GIFT-0` is the original `SUNDAE` scheme presented in the `SUNDAE` paper which does not require a nonce.

The members are arranged in the order of preference and potential use-case. The primary member, `SUNDAE-GIFT-96`, satisfies the requirements set by NIST. Next, member 2, `SUNDAE-GIFT-0`, is preferred as it removes the overhead for sending a nonce, which is non-negligible overhead cost for small messages. Subsequently, we believe that the use-case of 128-bit nonce is more common than 64-bit nonce, hence the ordering of `SUNDAE-GIFT-128` and `SUNDAE-GIFT-64`.

Modification of the existing mode and primitive There is sight modifications to `SUNDAE` (see Section 2.3) as compared to the specification in `SUNDAE` paper but only for the purpose to accommodate fixed length nonce as specified in the NIST requirement. There is no modification to the `GIFT-128` algorithm, apart from some update in the notations and a new perspective of viewing the incoming data (see Section 2.4 and 2.5).

2.3 Specification of SUNDAE

SUNDAE consists of an encryption algorithm and a decryption algorithm. It is parametrized by a block cipher, which in this document is GIFT-128 with a key set $\mathsf{K} := \{0,1\}^{128}$ and block size $n = 128$.The encryption algorithm enc takes as input a key $K \in \mathsf{K}$, an associated data $A \in \{0,1\}^*$, and a message $M \in \{0,1\}^*$. Depending on the variant, SUNDAE may accept a fixed length nonce N (details to follow) which is prepended to and regarded as part of the associated data A. Thus, for simplicity, we omit the notation N and let $A \leftarrow N\|A$. It outputs a ciphertext $C \in \{0,1\}^{n+|M|}$, where the first n bits of the ciphertext are interpreted as a *tag* T. The decryption algorithm dec takes as input a key $K \in \mathsf{K}$, an associated data $A \in \{0,1\}^*$, and a ciphertext $C \in \{0,1\}^n \times \{0,1\}^*$, and outputs $M \in \{0,1\}^{|C|-n}$, or the error symbol $\perp$ if verification is not successful. The encryption and decryption algorithms are such that for all $K \in \mathsf{K}$, $A \in \{0,1\}^*$, $M \in \{0,1\}^*$, with $|A| + |M| \geq 0$,

$$\mathsf{dec}_K\big(A, \mathsf{enc}_K(A, M)\big) = M\,.$$

The key $K \in \mathsf{K}$ is assumed uniformly at random from K. After fixing a key, uniqueness should be guaranteed of each pair (A, M) of associated data and message input; associated data can be repeated if the message is changed, and message input may be repeated if the associated data is changed. Caution must be taken so that intermediate values used during encryption and decryption are not leaked. In particular, unverified plaintext from the decryption algorithm should not be released [1].

Fig. 1 gives a diagram of encryption. All block cipher calls are performed with a fixed key K. Both encryption and decryption algorithms only use the "forward" block cipher E_K, hence the decryption algorithm of the block cipher is not required.

Encryption Algorithm The encryption algorithm performs the first pass of the associated data and message to produce the tag. After which, the tag is used to encrypt the message to produce the ciphertext.

Initialization. An initial block $B = b_{127}b_{126}b_{125}b_{124}\|0^{124}$ is defined depending on the variant and input parameters, where

$$b_{127} = \begin{cases} 0 & \text{if } |A| = 0 \\ 1 & \text{otherwise}\,. \end{cases}$$

$$b_{126} = \begin{cases} 0 & \text{if } |M| = 0 \\ 1 & \text{otherwise}\,. \end{cases}$$

$$b_{125}b_{124} = \begin{cases} 00 & \text{if } |N| = 0 \\ 01 & \text{if } |N| = 64 \\ 10 & \text{if } |N| = 96 \\ 11 & \text{if } |N| = 128\,. \end{cases}$$

Note that if the nonce is non-empty, $|N| > 0$, then naturally $b_{127} = 1$ as the nonce is part of the associated data.
Next, a block cipher call is made to produce the initial chaining value V.

$$V \leftarrow \mathsf{E}_K(B)$$

If there is no associated data and message, the encryption is completed and V is outputted as tag, $T = V$.

Processing associated data. If the associated data A is empty, this step is skipped and proceed to processing message directly. If A is non-empty, it is partitioned into n-bit blocks,

$$A[1]A[2]\cdots A[\ell_A] \xleftarrow{n} A\,.$$

For the first $\ell_A - 1$ blocks, it is XORed to V followed by another block cipher call.

$$V \leftarrow \mathsf{E}_K(V \oplus A[i]),\ \forall i \in \{1,\ldots,\ell_A - 1\}\,.$$

For the last block $A[\ell_A]$, it is first padded before XORing to V, and V is updated by some multiplication depending on the length of $A[\ell_A]$.

$$V \leftarrow \mathsf{E}_K(m \times (V \oplus \mathsf{pad}(A[\ell_A])))\,,\ \text{where } m = \begin{cases} 2 & \text{if } |A[\ell_A]| < n \\ 4 & \text{otherwise}\,. \end{cases}$$

If the message M is empty, then the encryption is completed and V is outputted as tag $T = V$ which is like a MAC for the associated data.

Processing message. The message M is partitioned into n-bit blocks,

$$M[1]M[2]\cdots M[\ell_M] \xleftarrow{n} M\,.$$

For the first $\ell_M - 1$ blocks, it is XORed to V followed by another block cipher call.

$$V \leftarrow \mathsf{E}_K(V \oplus M[i]),\ \forall i \in \{1,\ldots,\ell_M - 1\}\,.$$

For the last block $M[\ell_M]$, it is first padded before XORing to V, and V is updated by some multiplication depending on the length of $M[\ell_M]$.

$$V \leftarrow \mathsf{E}_K(m \times (V \oplus \mathsf{pad}(M[\ell_M])))\,,\ \text{where } m = \begin{cases} 2 & \text{if } |M[\ell_M]| < n \\ 4 & \text{otherwise}\,. \end{cases}$$

Extracting tag. The tag T is assigned with the chaining value V.

$$T \leftarrow V$$

Encrypting message. The message blocks are encrypted block by block without padding as follows

$$\begin{aligned} V &\leftarrow \mathsf{E}_K(V) \\ C[i] &\leftarrow \lfloor \mathsf{E}_K(V) \rfloor_{|M[i]|} \oplus M[i] \end{aligned} \qquad \forall i \in \{1,\ldots,\ell_M\}\,.$$

Finally, the tag and ciphertext blocks are concatenated and outputted as the ciphertext C.

$$C \leftarrow TC[1]C[2]\cdots C[\ell_M]$$

Decryption Algorithm The decryption algorithm first decrypts the ciphertext with the given tag before processing the associated data and decrypted message to produce another tag for verification.

Extracting tag. The ciphertext C is partitioned into n-bit blocks,

$$C[1]C[2]\cdots C[\ell_C] \xleftarrow{n} C\,.$$

The first block of C is taken as the chaining value $V \leftarrow C[1]$.

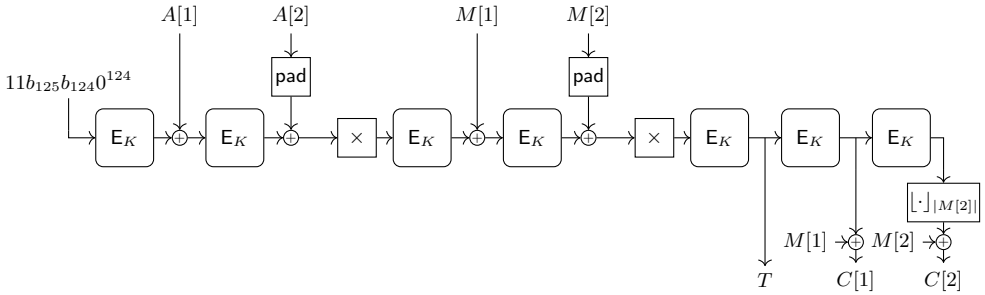
Decrypting message. If $|C| = n$, then this step is skipped and proceed to generating the tag. Otherwise, the decryption is similar to the encryption,

$$\begin{aligned} V &\leftarrow \mathsf{E}_K(V) \\ M[i-1] &\leftarrow \lfloor \mathsf{E}_K(V) \rfloor_{|C[i]|} \oplus C[i] \end{aligned} \qquad \forall i \in \{2, \ldots, \ell_C\} .$$

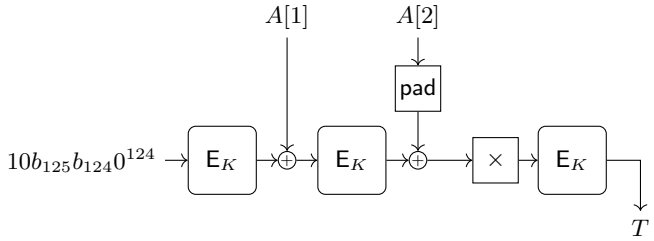
Generating tag. The exact same steps in the encryption algorithm, from "initialization" to "extracting tag", are performed to generate a new tag T' based on the associated data and decrypted message.

Verifying tag. If the new tag does not match the given tag, i.e. $T' \neq C[1]$, the verification fails and there will be null output $\perp$. Otherwise, the decrypted message blocks (if any) are concatenated and outputted as the decrypted message M.

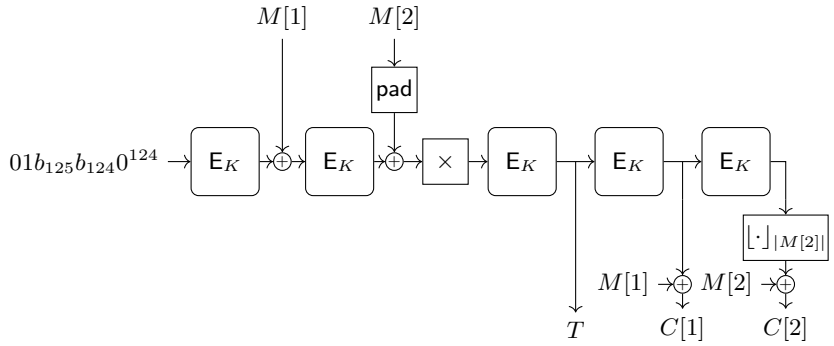
$$M \leftarrow \begin{cases} M[1] \cdots M[\ell_C - 1] & \text{if } |C| > n \\ \varepsilon & \text{otherwise} . \end{cases}$$



(a) `SUNDAE-GIFT` encryption with associated and plaintext data. The box below the rightmost block cipher call represents truncation.



(b) `SUNDAE-GIFT` without plaintext data, meaning only a tag is produced, like a MAC.



(c) `SUNDAE-GIFT` encryption with only plaintext data.

Fig. 1: Diagrams of `SUNDAE-GIFT` encryption and authentication. The initial block cipher call changes depending upon the presence of associated and plaintext data. $b_{125}b_{124}$ is defined based on the `SUNDAE-GIFT` member used. The multiplication $\times$ by 2 or 4 and depends on the length of the last blocks.

2.4 Specification of GIFT-128

GIFT-128 is an 128-bit Substitution-Permutation network (SPN) based block cipher with a key length of 128-bit. It has is a 40-round iterative block cipher with identical round function. There are two versions of GIFT, namely GIFT-64 and GIFT-128. But since we are focusing only on GIFT-128 in this document, we use GIFT and GIFT-128 interchangeably.

There are different ways to perceive GIFT-128, the more pictorial description is detailed in Section 2 of 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 2). In this document, we will be using bitslice description which is similar to Appendix A of the GIFT paper.

Round function. Each round of GIFT 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.

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

Refer to Section 2.5 for details of the incoming 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.

In Table 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.

Table 2: Specifications of `GIFT-128` bit permutation.

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

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,

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

where $\ggg i$ is an i bits right rotation within the 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 `SUNDAE-GIFT`.

2.5 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 incurs 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.

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

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

where B_i are the arriving bytes.

Relation to `GIFT-128` LUT implementation. An alternative implementation of `GIFT` is using look-up table (LUT) for the SubCells operation. Such implementation prefers having the data in the conventional format, i.e. $B_0 B_1 \cdots B15 = b_{127} b_{126} \cdots b_1 b_0$.

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

$$\begin{bmatrix} B_0 \parallel B_1 \parallel B_2 \parallel B_3 \\ B_4 \parallel B_5 \parallel B_6 \parallel B_7 \\ B_8 \parallel B_9 \parallel B_{10} \parallel B_{11} \\ B_{12} \parallel B_{13} \parallel B_{14} \parallel B_{15} \end{bmatrix} = \begin{bmatrix} b_{124} b_{120} b_{116} \cdots b_{96} \parallel b_{92} \cdots b_{64} \parallel b_{60} \cdots b_{32} \parallel b_{28} \cdots b_0 \\ b_{125} b_{121} b_{117} \cdots b_{97} \parallel b_{93} \cdots b_{65} \parallel b_{61} \cdots b_{33} \parallel b_{29} \cdots b_1 \\ b_{126} b_{122} b_{118} \cdots b_{98} \parallel b_{94} \cdots b_{66} \parallel b_{62} \cdots b_{34} \parallel b_{30} \cdots b_2 \\ b_{127} b_{123} b_{119} \cdots b_{99} \parallel b_{95} \cdots b_{67} \parallel b_{63} \cdots b_{35} \parallel b_{31} \cdots b_3 \end{bmatrix}.$$

First, unpack the data into the conventional format. Next, perform the LUT implementation of `GIFT`. Finally, pack the output data back to the bitslice format. No additional packing/unpacking is required for the key. This will yield the exact same bitslice implementation as we described in the Section 2.4.

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`

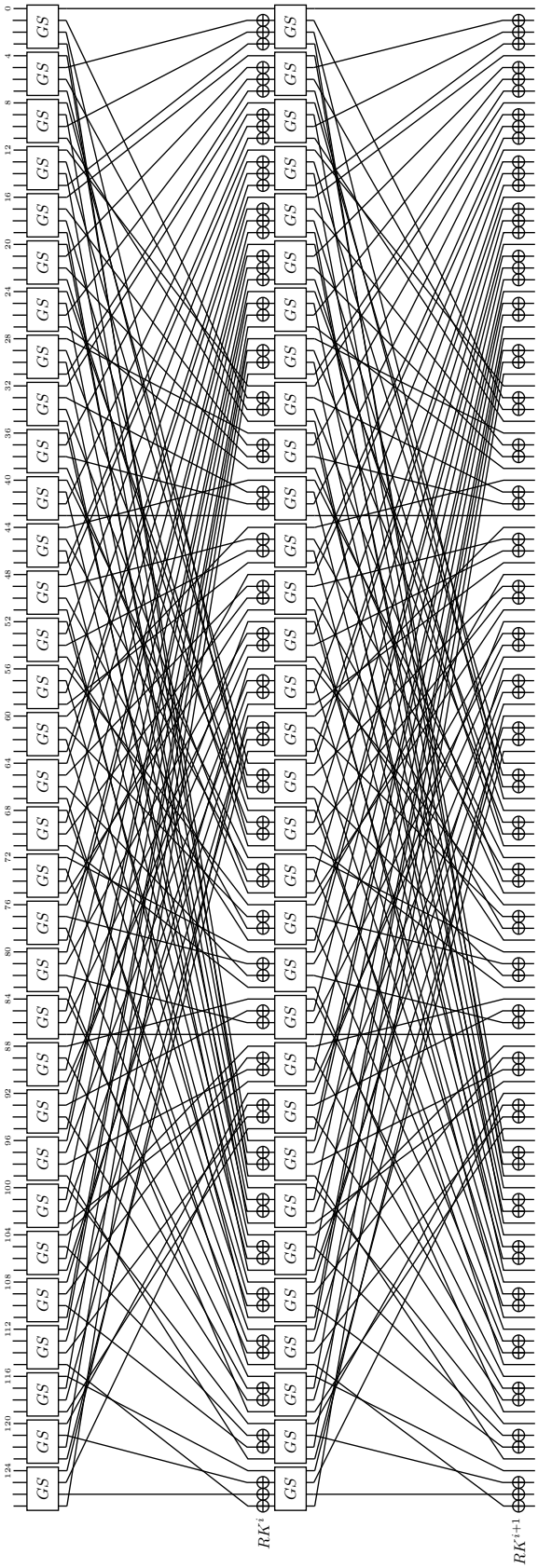


Fig. 2: 2 rounds of GIFT-128.

3 Security

3.1 Security Goals and Security Claims

Two security goals are listed by the NIST requirements document, which are expected to hold as long as the nonce is not repeated under the same key:

1. confidentiality of the plaintexts under adaptive chosen-plaintext attacks, and
2. integrity of the ciphertexts under adaptive forgery attempts.

We claim all SUNDAE-GIFT members satisfy both goals by achieving AE security as given by the nAE security definition of Namprempre et al. [28]. Note that one of the SUNDAE-GIFT family members does not have a nonce input — **we claim nAE security for the nonceless SUNDAE-GIFT-0 assuming the combination of plaintext and associated data does not repeat.**

Furthermore, all SUNDAE-GIFT members are designed as *deterministic AEAD schemes*, hence they achieve DAE security as defined in the SUNDAE paper. This means that if the combination of nonce, plaintext, and associated data is repeated, then only equality of plaintexts is leaked. In particular, integrity is always maintained, and confidentiality maintained for any unique combination of nonce, plaintext, and associated data.

We give our security claims for both nAE and DAE security relative to a mode confidence level ϵ and primitive confidence level δ, which are values between zero and one denoting the maximum probability that an adversary will be able to breach security of the mode and the primitive, respectively. The adversary's overall success probability is $\epsilon + \delta$.

We claim that adversaries will not be able to mount attacks against the primitive more successfully than generic attacks possible against any block cipher with the same key and block sizes. Then, assuming δ remains negligbly small (say, $\delta \leq 2^{-16}$) and adversaries do not capture or produce more than β bytes of plaintext, associated data, or ciphertext,

$$\epsilon \leq \frac{42}{256} \cdot \frac{\beta^2}{2^{128}} \,. \tag{1}$$

Note that the above equation represents a conservative upper bound. A less conservative relationship between ϵ and β can be computed using the original bound contained in the SUNDAE paper (where $\sigma = \beta/16$). Table 3 lists some bounds on the amount of data that the adversary can capture or produce (β), relative to values of ϵ[7].

Table 3: Minimum data limits per key that adversaries should produce or capture to achieve the given adversarial advantage.

Data Limit (Bytes)	β	2^{35}	2^{40}	2^{45}	2^{50}	2^{55}	2^{60}
Adversarial Advantage	ϵ	2^{-60}	2^{-50}	2^{-40}	2^{-30}	2^{-20}	2^{-10}

3.2 Security Rationale

Mode Security. For brevity's sake, we do not rewrite the security analysis and refer readers to in Section 4 of the SUNDAE paper. Assuming GIFT is secure as a pseudorandom permutation, SUNDAE's bounds hold.

[7] Note that SUNDAE-GIFT is not unique in having data limits relative to adversarial advantage: all modes of operation will have similar limits.

Primitive Security. The security analysis of GIFT-128 is provided in Section 4 of the GIFT paper. Here we highlight several important features.

Differential cryptanalysis. Zhu et al. applied the mixed-integer-linear-programming based differential characteristic search method for GIFT-128 and found an 18-round differential characteristic with probability 2^{-109} [34], which was further extended to a 23-round key recovery attack with complexity $(Data, Time, Memory) = (2^{120}, 2^{120}, 2^{80})$. We expect that full (40) rounds are secure against differential cryptanalysis.

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

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

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

4 Performance

4.1 Hardware Performance in ASIC

Lightweight implementations of CLOC, SILC [21] and AES-OTR [26], with AES-128 as the underlying block cipher has already been published in [5]. As with all rate 1/2 modes like CLOC, SILC, and also some rate 1 modes like OTR [5] there is a need for offline storage of message blocks for reading them twice. Note that this was also assumed in the lightweight implementation of the above modes in [5]. In this work, the authors use the 8-bit serial implementation of AES given in [27]. The authors implemented the above modes for two typical use cases (a) aggressive and (b) conservative. The aggressive design implemented a version of the circuit that only catered to a limited set of sizes of the plaintext and associated data. For example, the aggressive circuit was only designed to process user inputs in which the associated data was empty and the length of the plaintext was an integral multiple of the block size of the underlying block cipher. The intermediate outputs produced by circuit were stored offline, and an external processor made them available at the input buses as required by the design. This relaxed many of the storage requirements in the circuit, and so the circuit occupied lower gate area. The conservative circuit had no such constraints and was designed to handle all types of user inputs within certain bounds (upto 8 blocks of associated data and 256 blocks of plaintext). All outputs of intermediate modules were stored in additional registers in the circuit. As a result of which its gate area was significantly larger.

SUNDAE was implemented with both AES [18] and PRESENT [13] as the underlying block ciphers in [4]. Since the mode of operation was designed in a manner that did not require temporary storage of any intermediate results, as a result no additional storage elements are required for this purpose. For the AES based mode, the Atomic-AES architectures developed in [7, 8] were used. These were 8-bit serial architectures for AES meant for accommodating both encryption/decryption on the same platform. Furthermore, the requirement of an additional register to perform field doublings and quadruplings was done away with by changing the structure of the finite field. In stead of performing doubling

over GF(2^{128}), 8 doublings over GF(2^{16})/ $< x^{16} + x^5 + x^3 + x + 1 >$ was done in a way that could be easily accommodated in the Atomic-AES architecture.

However as noted in [6], in addition to low throughput, serial architectures do not have very good energy performance, as any basic operation like encryption/decryption takes a lot of clock cycles to execute. Thus keeping energy and speed aspects in mind, our primary implementation is based on the round based block cipher circuits. The `GIFT` block cipher was designed with a motivation for good performance on lightweight platforms. The roundkey additon for the cipher is over only half the state and the keyschedule being only a bit permutation does not require logic gates. These characteristics make the `GIFT` well suited for lightweight applications. In fact as reported in [9], among the block ciphers defined for 128-bit block size `GIFT-128` has the lowest hardware footprint and very low energy consumption.

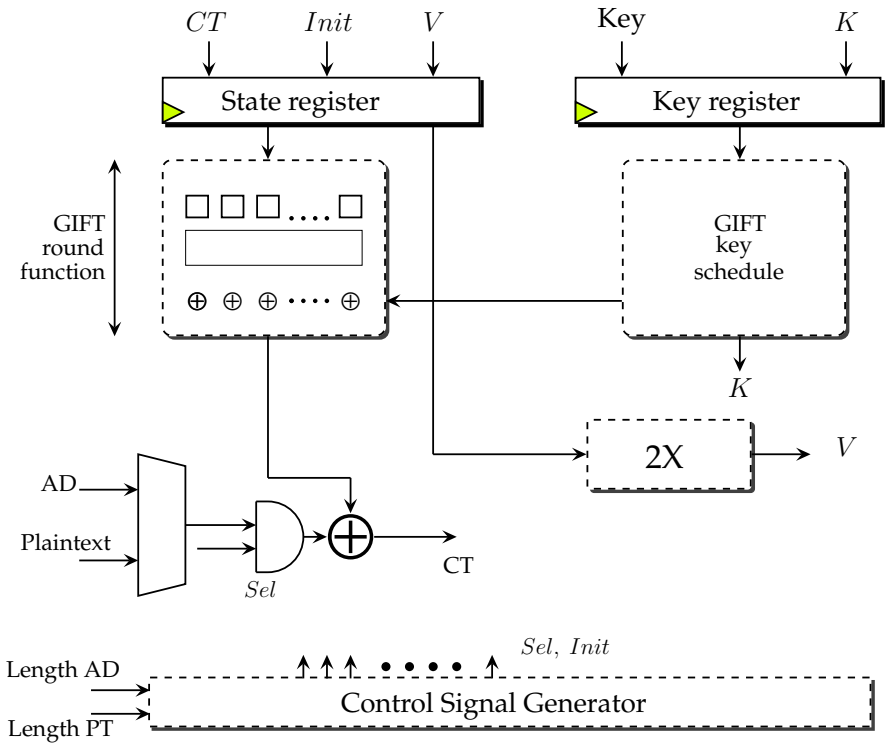


Fig. 3: Hardware circuit for round based `SUNDAE-GIFT`

Figure 3 details the hardware circuit for round based `SUNDAE-GIFT`. The mode is designed in a manner so as to not require any additional state bits apart from the ones used in the block cipher circuit. Thus the design only requires the state and key registers for storage. The initial input (denoted by *Init* 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 *Init*, output of the doubling circuit or the signal generated by adding the associated data/plaintext to the `GIFT-128` round function. Plaintext addition is not required in the second pass of the message during which the *Sel* signal controlling the and gate before addition is set to 0.

4.2 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 $Init$ signal. Each block of associated data would take E cycles to process. At the end of the associated data pass, $D_a = 1$ or 2 clock cycles are required for doubling or quadrupling according as the last associated data block is full length or not. Similar number of clock cycles are required in the first message pass, and an additional $D_m = 1$ or 2 cycles are required in case of doubling or quadrupling as required. Each block of plaintext in the final message pass, would again take E cycles. Hence the total number of cycles taken is equal to $T = E + n_a E + D_a + n_m E + D_m + n_m E$, where n_a, n_m are the total number of associated data/ message blocks.

4.3 Performance

In Table 4 we present the synthesis results for the designs. 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 SUNDAE-GIFT occupied 3494 GE. A component-wise breakup of the circuit is givem in Figure 4. In Table 4 we present detailed comparison of SUNDAE-GIFT with SUNDAE implemented with other block ciphers. We measure energy consumed at 10 Mhz for various lengths of associated data and plaintext. The 8 and 4 bit serial circuits although have a lower hardware footprint, they take more clock cycles to compute a unit encryption and hence are not energy efficient.

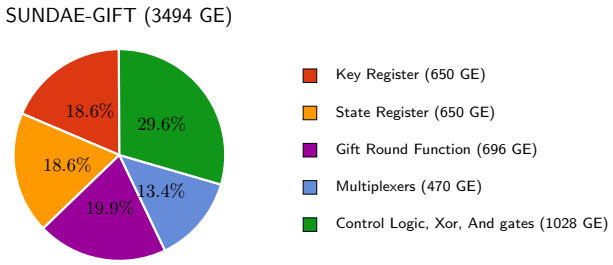


Fig. 4: Component-wise breakup of the SUNDAE-GIFT circuit

4.4 Performance in FPGA

A comprehensive study of implementation of the GIFT cipher on various FPGA platforms (of the Spartan 6 and Artix 7 families) was done in [23]. The authors reported three possible architectures of GIFT: round-based, and two types of serial architectures that operates using different widths of datapaths. The first serial architecture (referred to as Serial-1) uses 8-bit input/output ports for data loading/unloading and has a serialized

Block Cipher	Circuit	Area (GE)	Power(μW)	Energy(nJ)					
				AD	PT	AD	PT	AD	PT
				0B	16B	16B	16B	16B	32B
AES-128	8-bit serial	2524	126.1	9.3		12.3		18.1	
PRESENT-80	4-bit serial	1452	50.9	14.8		19.8		30.9	
GIFT-128	Round based	3494	170.1	**2.1**		**2.8**		**4.2**	

Table 4: Implementation results for SUNDAE. (Power reported at 10 MHz)

application of the substitution layer based on two 4-bit Sboxes. Thus the substitution layer of GIFT-128 would take 128/8=16 cycles in Serial-1. The permutation layer and key addition is performed in the 17th clock cycle much like the PRESENT architecture in [31], resulting in an encryption latency of $16 + 40 * 17 + 16 = 712$ cycles.

The second serial architecture (referred to as Serial-2) uses 32-bit input/output ports for data loading/unloading and has a serialized application of the substitution layer based on eight 4-bit. Sboxes. Thus 4 clock cycles are required for the substitution layer. This implementation takes advantage of the fact that the GIFT-128 permutation function can be written as the composition of a columnwise permutation and a transposition function. The strategy therefore is to compute at the same time the columnwise S-box, key addition and permutation functions in the 4 cycles allotted for the substitution function. The 5-th cycle is used for the matrix transposition operation, resulting in a 5-cycle round and a total latency of $4 + 5 * 40 + 4 = 208$ cycles.

In this submission we have tweaked slightly the format of the incoming data, and thus the implementations reported in [23] needs to be tweaked slightly, only for the Serial-1 and Serial-2 architectures. Namely we will need to spend one extra cycle per round to permute the arrangement of bits of the incoming bitslice format to the conventional format before applying round function operations. This incurs only a 40 cycle penalty in the encryption latency.

4.5 Threshold Implementation

The algebraic degree of the GIFT Sbox is 3 (same as PRESENT) and as such constructing threshold circuits is slightly more difficult than for quadratic Sboxes. However threshold implementations of the round-based GIFT-128 circuit has been extensively studied in [20]. Since the S-box is cubic, the number of direct shares it must be decomposed to needs to be at least 4. However, the authors in [20] report three philosophies.

The first decomposes the Sbox as the composition $F \circ G$ of two quadratic S-boxes F, G, and implements each decomposed Sbox using 3 shares with a register separating the two shared implementations, as in [29]. The shares of both G, F being algebraically similar to each other, and differing only in the order of input bits, the authors further apply an optimization due to [22], that reduces the area of the circuit by implementing the shares over 3 cycles, using a multiplexer to permute the order of bits each time.

The second is a direct sharing approach using 4 shares, and a third approach proposed by them uses only 3 shares for strictly resource-constrained platforms. In total the authors propose 9 different threshold circuits for GIFT-128, depending on whether the key is shared or not, and the type of circuit optimization used. The circuits were synthesized using the TSMC low power 65 nm standard cell library. The smallest implementation uses 3 shares and 256 random bits and occupies around 13349 GE and is around 5.38 times the size of the unprotected circuit. The largest implementation uses 4 direct shares for both the key and datapath and occupies around 94 kGE. For more results, we refer the reader to [20].

5 Features and Design Rationales

We highlight the main features of SUNDAE-GIFT:

- **Deterministic authenticated encryption:** The design does not depend on the randomness or uniqueness of the nonce, providing maximal robustness to a lack of proper randomness or secure state.
- **Proven secure:** The security of the SUNDAE mode of operation is proven secure relative to its underlying block cipher.
- **Small area:** The state size of SUNDAE is optimal as it only requires the minimum of one single n-bit state. The underlying primitive GIFT-128 is one of, if not the, most lightweight 128-bit block ciphers. Therefore, SUNDAE-GIFT is possibly the most lightweight construction (in terms of hardware area) that one would achieve for a block cipher based AEAD scheme.
- **Efficient on short messages:** SUNDAE-GIFT is particularly suitable for short messages, which are typical in many deployment scenarios for lightweight cryptography.
- **High performance in hardware and software:** While offering high performance in hardware implementations, SUNDAE-GIFT can also be implemented very efficiently in a bitsliced manner for high-performance software environments, making full use of SIMD instructions such as AVX2 or AVX-512 on Intel platforms. It is thus equally suitable for deployment on resource-constrained devices and the server side.

A downside of SUNDAE-GIFT is that it requires a two-pass of the message, alternatively perceives as rate 0.5. But we argue that it is acceptable for small messages.

Rationale for Bitslicing GIFT-128. For brevity sake, we do not rewrite the design rationale of GIFT-128 and refer readers to in Section 3 of the GIFT paper. In short, the 4-bit Sbox of GIFT-128 is extremely lightweight, but has the drawback of having branching number 2 only. This issue was taken care of by carefully choosing the linear layer of GIFT-128, which actually corrected the well-known linear cryptanalysis weaknesses of PRESENT. The linear layer of GIFT-128 is basically for free in a round-based hardware implementation (being composed of bit-wiring), and the same can be said for the key schedule that consists of simple shifts. Finally, the constant are generated with a very lightweight LFSR.

We now explain the choice of using bitslice implementation of GIFT-128. 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 Sboxes (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 Sboxes 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 Sboxes 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 is extremely slow and complex to implement. This effect can be reduced by doing several blocks in parallel using none other than bitslice implementation. Even for a single block encryption, bitslice implementation is still more efficient that LUT based implementation because of the way the bits are packed.

Third and lastly the key addition, for LUT based implementation, the subkeys need to be XORed to bit positions that are 3 bits apart, making the key addition tedious and

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

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). The authors would like to thank Atul Luykx for the meaningful discussions with regards to this submission.

References

1. Andreeva, E., Bogdanov, A., Luykx, A., Mennink, B., Mouha, N., Yasuda, K.: How to securely release unverified plaintext in authenticated encryption. In Sarkar, P., Iwata, T., eds.: Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I. Volume 8873 of Lecture Notes in Computer Science., Springer (2014) 105–125
2. Andreeva, E., Bogdanov, A., Luykx, A., Mennink, B., Tischhauser, E., Yasuda, K.: Parallelizable and authenticated online ciphers. In Sako, K., Sarkar, P., eds.: Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part I. Volume 8269 of Lecture Notes in Computer Science., Springer (2013) 424–443
3. Banik, S., Bogdanov, A., Isobe, T., Shibutani, K., Hiwatari, H., Akishita, T., Regazzoni, F.: Midori: A block cipher for low energy. In Iwata, T., Cheon, J.H., eds.: Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II. Volume 9453 of Lecture Notes in Computer Science., Springer (2015) 411–436
4. Banik, S., Bogdanov, A., Luykx, A., Tischhauser, E.: Sundae: Small universal deterministic authenticated encryption for the internet of things. IACR Transactions on Symmetric Cryptology **2018**(3) (Sep. 2018) 1–35
5. Banik, S., Bogdanov, A., Minematsu, K.: Low-area hardware implementations of cloc, SILC and AES-OTR. In Robinson, W.H., Bhunia, S., Kastner, R., eds.: 2016 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2016, McLean, VA, USA, May 3-5, 2016, IEEE Computer Society (2016) 71–74
6. Banik, S., Bogdanov, A., Regazzoni, F.: Exploring energy efficiency of lightweight block ciphers. In: Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers. (2015) 178–194
7. Banik, S., Bogdanov, A., Regazzoni, F.: Atomic-aes: A compact implementation of the AES encryption/decryption core. In Dunkelman, O., Sanadhya, S.K., eds.: Progress in Cryptology - INDOCRYPT 2016 - 17th International Conference on Cryptology in India, Kolkata, India, December 11-14, 2016, Proceedings. Volume 10095 of Lecture Notes in Computer Science. (2016) 173–190
8. Banik, S., Bogdanov, A., Regazzoni, F.: Atomic-aes v 2.0. IACR Cryptology ePrint Archive **2016** (2016) 1005
9. 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: Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. (2017) 321–345
10. Banik, S., Pandey, S.K., Peyrin, T., Sim, S.M., Todo, Y., Sasaki, Y.: Gift: A small present. Cryptology ePrint Archive, Report 2017/622 (2017) https://eprint.iacr.org/2017/622.

11. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In Robshaw, M., Katz, J., eds.: Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II. Volume 9815 of Lecture Notes in Computer Science., Springer (2016) 123–153
12. Bellare, M., Rogaway, P., Wagner, D.: The EAX mode of operation. In Roy, B.K., Meier, W., eds.: Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers. Volume 3017 of Lecture Notes in Computer Science., Springer (2004) 389–407
13. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: an ultra-lightweight block cipher. In Paillier, P., Verbauwhede, I., eds.: Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings. Volume 4727 of Lecture Notes in Computer Science., Springer (2007) 450–466
14. Bogdanov, A., Mendel, F., Regazzoni, F., Rijmen, V., Tischhauser, E.: ALE: aes-based lightweight authenticated encryption. In Moriai, S., ed.: Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers. Volume 8424 of Lecture Notes in Computer Science., Springer (2013) 447–466
15. Cannière, C.D., Dunkelman, O., Knezevic, M.: KATAN and KTANTAN - A family of small and efficient hardware-oriented block ciphers. In Clavier, C., Gaj, K., eds.: Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings. Volume 5747 of Lecture Notes in Computer Science., Springer (2009) 272–288
16. Chakraborti, A., Iwata, T., Minematsu, K., Nandi, M.: Blockcipher-based authenticated encryption: How small can we go? In Fischer, W., Homma, N., eds.: Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. Lecture Notes in Computer Science, Springer (2017) 21 To appear.
17. Cid, C., Rechberger, C., eds.: Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers. Volume 8540 of Lecture Notes in Computer Science., Springer (2015)
18. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography. Springer (2002)
19. Gueron, S., Lindell, Y.: GCM-SIV: full nonce misuse-resistant authenticated encryption at under one cycle per byte. In Ray, I., Li, N., Kruegel, C., eds.: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015, ACM (2015) 109–119
20. Gupta, N., Jati, A., Chattopadhyay, A., Sanadhya, S.K., Chang, D.: Threshold implementations of GIFT: A trade-off analysis. IACR Cryptology ePrint Archive **2017** (2017) 1040
21. Iwata, T., Minematsu, K., Guo, J., Morioka, S.: CLOC: authenticated encryption for short input. [17] 149–167
22. Kutzner, S., Nguyen, P.H., Poschmann, A., Wang, H.: On 3-share threshold implementations for 4-bit s-boxes. In: Constructive Side-Channel Analysis and Secure Design - 4th International Workshop, COSADE 2013, Paris, France, March 6-8, 2013, Revised Selected Papers. (2013) 99–113
23. Lara-Nino, C.A., Diaz-Perez, A., Morales-Sandoval, M.: Fpga-based assessment of midori and gift lightweight block ciphers. In: Information and Communications Security - 20th International Conference, ICICS 2018, Lille, France, October 29-31, 2018, Proceedings. (2018) 745–755
24. Luykx, A., Preneel, B., Tischhauser, E., Yasuda, K.: A MAC mode for lightweight block ciphers. In Peyrin, T., ed.: Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers. Volume 9783 of Lecture Notes in Computer Science., Springer (2016) 43–59
25. Matsuda, S., Moriai, S.: Lightweight cryptography for the cloud: Exploit the power of bitslice implementation. In Prouff, E., Schaumont, P., eds.: Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings. Volume 7428 of Lecture Notes in Computer Science., Springer (2012) 408–425

26. Minematsu, K.: Parallelizable rate-1 authenticated encryption from pseudorandom functions. In Nguyen, P.Q., Oswald, E., eds.: Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings. Volume 8441 of Lecture Notes in Computer Science., Springer (2014) 275–292
27. Moradi, A., Poschmann, A., Ling, S., Paar, C., Wang, H.: Pushing the limits: A very compact and a threshold implementation of AES. In Paterson, K.G., ed.: Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings. Volume 6632 of Lecture Notes in Computer Science., Springer (2011) 69–88
28. Namprempre, C., Rogaway, P., Shrimpton, T.: Reconsidering generic composition. Cryptology ePrint Archive, Report 2014/206 (2014) https://eprint.iacr.org/2014/206.
29. Poschmann, A., Moradi, A., Khoo, K., Lim, C., Wang, H., Ling, S.: Side-channel resistant crypto for less than 2, 300 GE. J. Cryptology **24**(2) (2011) 322–345
30. Rogaway, P., Shrimpton, T.: A provable-security treatment of the key-wrap problem. In Vaudenay, S., ed.: Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings. Volume 4004 of Lecture Notes in Computer Science., Springer (2006) 373–390
31. Rolfes, C., Poschmann, A., Leander, G., Paar, C.: Ultra-Lightweight Implementations for Smart Devices - Security for 1000 Gate Equivalents. In Grimaud, G., Standaert, F.X., eds.: CARDIS. Volume 5189 of Lecture Notes in Computer Science., Springer (2008) 89–103
32. Sasaki, Y.: Integer linear programming for three-subset meet-in-the-middle attacks: Application to gift. In Inomata, A., Yasuda, K., eds.: Advances in Information and Computer Security, Cham, Springer International Publishing (2018) 227–243
33. Wu, H., Huang, T.: The JAMBU Lightweight Authentication Encryption Mode (v2.1). CAESAR submissions (2016) https://competitions.cr.yp.to/round3/jambuv21.pdf.
34. Zhu, B., Dong, X., Yu, H.: Milp-based differential attack on round-reduced gift. Cryptology ePrint Archive, Report 2018/390 (2018) https://eprint.iacr.org/2018/390.

Changelog

- 29-03-2019: version v1.0