GIFT: A Small Present

Towards Reaching the Limit of Lightweight Encryption

Subhadeep Banik[1,5(✉)], Sumit Kumar Pandey[2], Thomas Peyrin[1,2,3], Yu Sasaki[3], Siang Meng Sim[2], and Yosuke Todo[4]

[1] Temasek Laboratories, Nanyang Technological University, Singapore, Singapore
{bsubhadeep,thomas.peyrin}@ntu.edu.sg
[2] School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore, Singapore
emailpandey@gmail.com, SSIM011@e.ntu.edu.sg
[3] School of Computer Science and Engineering, Nanyang Technological University, Singapore, Singapore
Sasaki.Yu@lab.ntt.co.jp
[4] NTT Secure Platform Laboratories, Tokyo, Japan
Todo.Yosuke@lab.ntt.co.jp
[5] LASEC, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland

Abstract. In this article, we revisit the design strategy of PRESENT, leveraging all the advances provided by the research community in construction and cryptanalysis since its publication, to push the design up to its limits. We obtain an improved version, named GIFT, that provides a much increased efficiency in all domains (smaller and faster), while correcting the well-known weakness of PRESENT with regards to linear hulls.

GIFT is a very simple and clean design that outperforms even SIMON or SKINNY for round-based implementations, making it one of the most energy efficient ciphers as of today. It reaches a point where almost the entire implementation area is taken by the storage and the Sboxes, where any cheaper choice of Sbox would lead to a very weak proposal. In essence, GIFT is composed of only Sbox and bit-wiring, but its natural bitslice data flow ensures excellent performances in all scenarios, from area-optimised hardware implementations to very fast software implementation on high-end platforms.

We conducted a thorough analysis of our design with regards to state-of-the-art cryptanalysis, and we provide strong bounds with regards to differential/linear attacks.

Keywords: Lightweight cryptography · Block cipher · PRESENT · GIFT

1 Introduction

In the past decade, the development of ubiquitous computing applications triggered the rapid expansion of the lightweight cryptography research field. All these applications operating in very constrained devices may require certain

W. Fischer and N. Homma (Eds.): CHES 2017, LNCS 10529, pp. 321–345, 2017.
DOI: 10.1007/978-3-319-66787-4_16

symmetric-key cryptography components to guarantee privacy and/or authentication for the users, such as block or stream ciphers, hash functions or MACs. Existing cryptography standards such as AES [18] or SHA-2 [33] are not always suitable for these strong constraints. There have been extensive research conducted in this direction, with countless new primitives being introduced [2,4,5,12,15,22,39], many of them getting broken rather rapidly (designing a cipher with strong constraints is not an easy task). Conforming to general trend, the American National Institute for Science and Technology (NIST) recently announced that it will consider standardizing some lightweight functions in a few years [34]. Some lightweight algorithms such as PRESENT [12], PHOTON [21] and SPONGENT [11] have already been included into ISO standards (ISO/IEC 29192-2:2012 and ISO/IEC 29192-5:2016).

Comparing different lightweight primitives is a very complex task. First, lightweight encryption encompasses a broad range of use cases, from passive RFID tags (that require a very low power consumption to operate) to battery powered devices (that require a very low energy consumption to maximise its life span) or low-latency applications (for disk encryption). While it is generally admitted that a major criterion for lightweight encryption is area minimisation, the throughput/area ratio is also very important because it shows the ability of the algorithm to provide good implementation trade-offs (this ratio is also correlated to the power or energy consumption of the algorithm). Moreover, the range of the various platforms to consider is very broad, starting from tiny RFID tags to rather powerful ARM processors. Even high-end servers have to be taken into account as it is likely that these very small and constrained devices will be communicating with back-end servers [6].

While most ciphers take lightweight hardware implementations into account to some extend, PRESENT [12] is probably one of the first candidates that was exclusively designed for that purpose. Its design is inspired by SERPENT [7] and is very simple: the round function is simply composed of a layer of small 4-bit Sboxes, followed by a bit permutation layer (essentially free in hardware) and a subkey addition. PRESENT has been extensively analysed in the past decade, and while its security margin has eroded, it remains a secure cipher. One can note that the weak point of PRESENT is the tendency of linear trails to cluster and to create powerful linear hulls [10,17].

Since the publication of PRESENT, many advances have been obtained, both in terms of security analysis and primitive design. The NSA proposed in 2013 two ciphers [4], SIMON and SPECK, that can reach much better efficiency in both hardware and software when compared to all other ciphers. However, this comes at the cost that proving simple linear/differential bounds for SIMON is much more complicated than for Substitution-Permutation-Network (SPN) ciphers like PRESENT (SIMON is based on a Feistel construction, with an internal function that uses only a AND, some XORs and some rotations). Besides, no preliminary analysis or rationale was provided by the SIMON authors. Last year, the tweakable block cipher SKINNY [5] was published to compete with SIMON's efficiency for round-based implementations, while providing strong linear/differential bounds.

As of today, SIMON and SKINNY seem to have a clear advantage in terms of efficiency when compared to other designs. Yet, PRESENT remains an elegant design, that suffers from being one of the first lightweight encryption algorithm to have been published, and thus not benefiting from the many advances obtained by the research community in the recent years.

Our Contributions. In this article, we revisit the PRESENT construction, 10 years after the original publication of PRESENT. This led to the creation of GIFT, a new lightweight block cipher, improving over PRESENT in both security and efficiency. Interestingly, our cipher 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, 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 90 nm Standard cell library.

	Area (GE)	Delay (ns)	Cycles	TP_{MAX} (MBit/s)	Power (μW) (@10 MHz)	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

In more details, we have revisited the PRESENT design strategy and pushed it to its limits, while providing special care to the known weak point of PRESENT: the linear hulls. The diffusion layer of PRESENT being composed of only a bit permutation, most of the security of PRESENT relies on its Sbox. This Sbox presents excellent cryptographic properties, but is quite costly. Indeed, it is trivial to see that the PRESENT Sbox needs to have a branching number of 3, or very good differential paths would exist otherwise (with only a single active Sbox per round). We managed to remove this constraint by carefully crafting the bit permutation in conjunction with the Difference Distribution Table (DDT)/Linear Approximation Table (LAT) of the Sbox. We remark that, to the best of the authors knowledge, this is the first time that the linear layer and the Sbox are fully intricate in a SPN cipher.

In terms of performances, removing this Sbox constraint allowed us to choose a much cheaper Sbox, which is actually what composes most of the overall area cost in PRESENT. GIFT is not only much smaller, but also much faster than PRESENT. As can be seen in Table 2, GIFT is by far the cipher that uses the least total number of operation per bit up to now. In terms of security, we are able to provide strong security bounds for simple differential and linear attacks. We can even show that GIFT is very resistant against linear hulls, and the clustering effect is greatly reduced when compared to PRESENT, thus correcting its main weak point. We have conducted a thorough security analysis of our candidate with state-of-the-art cryptanalysis techniques.

Table 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.	nb. of op.	round-based
		int. cipher	key sch.	total	w/o key sch.	w/key sch.	impl. area
GIFT-64-128	28	1 N		1 N	$3 \times 28 = 84$	$3 \times 28 = \mathbf{84}$	$1 + 2.67 \times 2 = \mathbf{6.34}$
		2 X		2 X			
SKINNY-64-128	36	1 N		1 N	$3.25 \times 36 = 117$	$3.875 \times 36 = \mathbf{139.5}$	$1 + 2.67 \times 2.875$
		2.25 X	0.625 X	2.875 X			
SIMON-64/128	44	0.5 A		0.5 A	$2 \times 44 = 88$	$3.5 \times 44 = \mathbf{154}$	$0.67 + 2.67 \times 3 = \mathbf{8.68}$
		1.5 X	1.5 X	3.0 X			
PRESENT-128	31	1 A	0.125 A	1.125 A	$4.75 \times 31 = 147.2$	$5.22 \times 31 = \mathbf{161.8}$	$1.5 + 2.67 \times 4.094$
		3.75 X	0.344 X	4.094 X			
GIFT-128-128	40	1 N		1 N	$3.0 \times 40 = 120$	$3.0 \times 40 = \mathbf{120}$	$1 + 2.67 \times 2 = \mathbf{6.34}$
		2 X		2 X			
SKINNY-128-128	40	1 N		1 N	$3.25 \times 40 = 130$	$3.25 \times 40 = \mathbf{130}$	$1 + 2.67 \times 2.25 = \mathbf{7.01}$
		2.25 X		2.25 X			
SIMON-128/128	68	0.5 A		0.5 A	$2 \times 68 = 136$	$3 \times 68 = \mathbf{204}$	$0.67 + 2.67 \times 2.5 = \mathbf{7.34}$
		1.5 X	1 X	2.5 X			
AES-128	10	4.25 A	1.06 A	5.31 A	$20.25 \times 10 = 202.5$	$24.81 \times 10 = \mathbf{248.1}$	$7.06 + 2.67 \times 19.5$
		16 X	3.5 X	19.5 X			

We end up with a very natural and clean cipher, with a simple round function and key schedule (composed of only a bit permutation, thus essentially free in hardware). The cipher can be seen in three different representations (classical 1D, bitslice 2D, and 3D), each offering simple yet different perspective on the cipher's security and opportunities for implementation improvements. GIFT comes in two versions, both with a 128-bit key: one 64-bit block version GIFT-64 and one 128-bit block version GIFT-128. The only difference between these two versions is the bit permutation to accommodate twice more state bits for GIFT-128.

In our hardware implementations of GIFT the storage composes about 75% of the total area, and the (very cheap) Sbox about 20%. Since any weaker choice of the Sbox would lead to a very insecure design, we argue that GIFT is probably very close to reaching the area limit of lightweight encryption.

Outline. We first specify GIFT in Sect. 2, and we provide the design rationale in Sect. 3. A thorough security analysis is performed in Sect. 4, while performances and implementation strategies are given in Sects. 5 and 6 for hardware and software respectively. All details are provided in the full version of the paper.

2 Specifications

In this work, we propose two versions of GIFT, GIFT-64-128 is a 28-round SPN cipher and GIFT-128-128 is a 40-round SPN cipher, both versions have a key length of 128-bit. For short, we call them GIFT-64 and GIFT-128 respectively.

GIFT can be perceived in three different representations. In this paper, we adopt the classical 1D representation, describing the bits in a row like PRESENT. It can also be described in bitslice 2D, a rectangular array like RECTANGLE [44], and even in 3D cuboid like 3D [32]. These alternative representations are detailed in the full version.

Round Function. Each round of GIFT consists of 3 steps: SubCells, PermBits, and AddRoundKey, which is conceptually similar to wrapping a gift:

1. Put the content into a box (SubCells);
2. Wrap the ribbon around the box (PermBits);
3. Tie a knot to secure the content (AddRoundKey).

Figure 1 illustrates 2 rounds of GIFT-64.

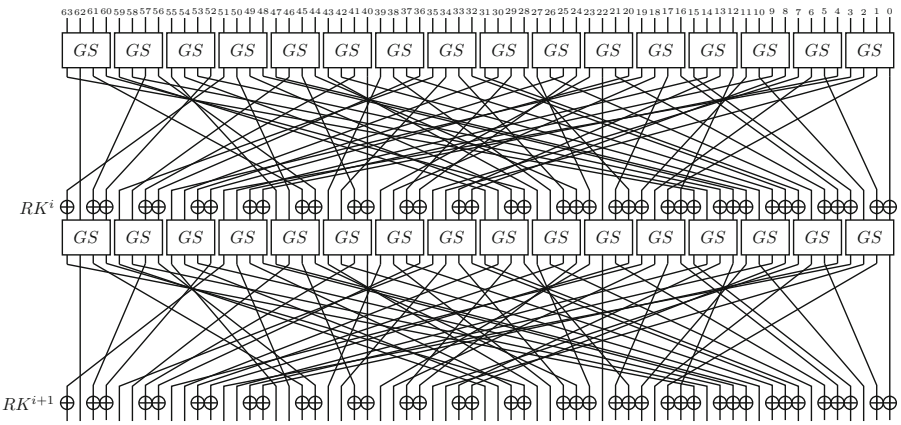

Fig. 1. 2 Rounds of GIFT-64.

Table 3. Specifications of GIFT Sbox GS.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$GS(x)$	1	a	4	c	6	f	3	9	2	d	b	7	5	0	8	e

Initialization. The cipher receives an n-bit plaintext $b_{n-1}b_{n-2}...b_0$ as the cipher state S, where $n = 64, 128$ and b_0 being the least significant bit. The cipher state can also be expressed as s many 4-bit nibbles $S = w_{s-1}||w_{s-2}||...||w_0$, where $s = 16, 32$. The cipher also receives a 128-bit key $K = k_7||k_6||...||k_0$ as the key state, where k_i is a 16-bit word.

SubCells. Both versions of GIFT use the same invertible 4-bit Sbox, GS. The Sbox is applied to every nibble of the cipher state. $w_i \leftarrow GS(w_i),\ \forall i \in \{0, ..., s-1\}$. The action of this Sbox in hexadecimal notation is given in Table 3.

PermBits. The bit permutation used in GIFT-64 and GIFT-128 are given in Tables 4 and 5 respectively. It maps bits from bit position i of the cipher state to bit position $P(i)$. $b_{P(i)} \leftarrow b_i,\ \forall i \in \{0, ..., n-1\}$.

AddRoundKey. This step consists of adding the round key and round constants. An $n/2$-bit round key RK is extracted from the key state, it is further partitioned into 2 s-bit words $RK = U||V = u_{s-1}...u_0||v_{s-1}...v_0$, where $s = 16, 32$ for GIFT-64 and GIFT-128 respectively.

For GIFT-64, U and V are XORed to $\{b_{4i+1}\}$ and $\{b_{4i}\}$ of the cipher state respectively. $b_{4i+1} \leftarrow b_{4i+1} \oplus u_i,\ b_{4i} \leftarrow b_{4i} \oplus v_i,\ \forall i \in \{0, ..., 15\}$.

For GIFT-128, U and V are XORed to $\{b_{4i+2}\}$ and $\{b_{4i+1}\}$ of the cipher state respectively. $b_{4i+2} \leftarrow b_{4i+2} \oplus u_i,\ b_{4i+1} \leftarrow b_{4i+1} \oplus v_i,\ \forall i \in \{0, ..., 31\}$.

For both versions of GIFT, a single bit "1" and a 6-bit round constant $C = c_5c_4c_3c_2c_1c_0$ are XORed into the cipher state at bit position $n-1$, 23, 19, 15, 11, 7 and 3 respectively. $b_{n-1} \leftarrow b_{n-1} \oplus 1,\ b_{23} \leftarrow b_{23} \oplus c_5,\ b_{19} \leftarrow b_{19} \oplus c_4,\ b_{15} \leftarrow b_{15} \oplus c_3,\ b_{11} \leftarrow b_{11} \oplus c_2,\ b_7 \leftarrow b_7 \oplus c_1,\ b_3 \leftarrow b_3 \oplus c_0$.

Table 4. Specifications of GIFT-64 Bit Permutation.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P_{64}(i)$	0	17	34	51	48	1	18	35	32	49	2	19	16	33	50	3
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P_{64}(i)$	4	21	38	55	52	5	22	39	36	53	6	23	20	37	54	7
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P_{64}(i)$	8	25	42	59	56	9	26	43	40	57	10	27	24	41	58	11
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P_{64}(i)$	12	29	46	63	60	13	30	47	44	61	14	31	28	45	62	15

Table 5. Specifications of GIFT-128 Bit Permutation.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P_{128}(i)$	0	33	66	99	96	1	34	67	64	97	2	35	32	65	98	3
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P_{128}(i)$	4	37	70	103	100	5	38	71	68	101	6	39	36	69	102	7
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P_{128}(i)$	8	41	74	107	104	9	42	75	72	105	10	43	40	73	106	11
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P_{128}(i)$	12	45	78	111	108	13	46	79	76	109	14	47	44	77	110	15
i	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
$P_{128}(i)$	16	49	82	115	112	17	50	83	80	113	18	51	48	81	114	19
i	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
$P_{128}(i)$	20	53	86	119	116	21	54	87	84	117	22	55	52	85	118	23
i	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
$P_{128}(i)$	24	57	90	123	120	25	58	91	88	121	26	59	56	89	122	27
i	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
$P_{128}(i)$	28	61	94	127	124	29	62	95	92	125	30	63	60	93	126	31

Key Schedule and Round Constants. The key schedule and round constants are the same for both versions of GIFT, the only difference is the round key extraction. A round key is *first* extracted from the key state before the key state update.

For GIFT-64, two 16-bit words of the key state are extracted as the round key $RK = U||V$. $U \leftarrow k_1,\ V \leftarrow k_0$.

For GIFT-128, four 16-bit words of the key state are extracted as the round key $RK = U||V$. $U \leftarrow k_5||k_4,\ V \leftarrow k_1||k_0$.

The key state is then updated as follows, $k_7||k_6||...||k_1||k_0 \leftarrow k_1 \ggg 2||k_0 \ggg 12||...||k_3||k_2$, where $\ggg i$ is an i bits right rotation within a 16-bit word.

The round constants are generated using the same 6-bit affine LFSR as SKINNY, whose state is denoted as $(c_5, c_4, c_3, c_2, c_1, c_0)$. 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

Remark: GIFT aims at single-key security, so we do not claim any related-key security (even though no attack is known in this model as of today). In case one wants to protect against related-key attacks as well, we advice to double the number of rounds.

3 Design Rationale

First, let us propose a subclassification for SPN ciphers.

Definition 1. *Substitution-bitPermutation network (SbPN) is a subclassification of Substitution-Permutation network, where the permutation layer (p-layer) only comprises of bit permutation. An m/n-SbPN cipher is an n-bit cipher in which substitution layer (s-layer) comprises of m-bit (Super-)Sboxes.*

For SPN ciphers like AES and SKINNY, we can shift the XOR components from the p-layer to the s-layer to form Super-Sboxes, leaving the p-layer with only bit permutation. For example, PRESENT is a 4/64-SbPN cipher, SKINNY-64 is a 16/64-SbPN cipher, and SKINNY-128 and AES are 32/128-SbPN ciphers.

Having that said, GIFT-64 is a 4/64-SbPN cipher while GIFT-128 is (probably the first of its kind) a 4/128-SbPN cipher.

3.1 The Designing of GIFT

Before we discuss the design rationale of GIFT, we would like to share some background story about GIFT, its design approach, and its comparison with another PRESENT-like ciphers.

The Origin of GIFT. It all started with a casual remark "What if the Sboxes in PRESENT are replaced with some smaller Sboxes, say the PICCOLO Sbox? It will be extremely lightweight since the core cipher only has some Sboxes and nothing else...". We quickly tested it but only to realise that the differential bounds became very low because the Sbox does not have differential branching number of 3. That is when we started analyzing the differential characteristics and studying the interaction between the linear layer and the Sbox. Surprisingly, we found that by carefully crafting the linear layer based on the properties of the Sbox, we were able to achieve the same differential bound as PRESENT without the constraint of differential branching number of 3. In addition, this result can also be applied to the improve linear cryptanalysis resistance which was lacking in PRESENT. Eventually, a small present—GIFT was created.

Design Approach. It is natural to ask how GIFT is different from the other lightweight primitives, especially the recent SKINNY family of block ciphers that was proposed at CRYPTO2016. One of the main difference is the design approach. SKINNY was designed with a high-security-reduce-area approach, that is to have a strong security property, then try to remove/reduce various components as much as possible. While GIFT adopts a small-area-increase-security approach, starting from a small area goal, we try to improve its security as much as possible.

Other PRESENT-like Ciphers. Besides PRESENT, one may also compare GIFT-64 with RECTANGLE since both are 4/64-SbPN ciphers and an improvement on the design of PRESENT. RECTANGLE was designed to be software friendly and to achieve a better resistance against the linear cryptanalysis as compared to PRESENT. However, although its bit permutation (ShiftRow) was designed to be software friendly, little analysis was done on the how differential and linear characteristics propagate through the cipher. Whereas for GIFT, we study the interplay of the Sbox and the bit permutation to achieve better differential and linear bounds. In addition, the ShiftRow of RECTANGLE achieves full diffusion in 4 rounds at best. Whereas GIFT-64 achieves full diffusion in 3 rounds like PRESENT, which can be proven to be the optimal for 4/64-SbPN ciphers.

3.2 Designing of GIFT Bit Permutation

To better understand the design rationale of the linear layer, we first look at the permutation layer of PRESENT to analyze the issue when the Sbox is replaced with another Sbox that does not have branching number of 3. Next, we show how we can solve this issue by carefully designing the bit permutation.

Linear Layer of PRESENT. The bit permutation of PRESENT is given in Table 6.

Table 6. Bit permutation of PRESENT.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P(i)$	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P(i)$	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P(i)$	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P(i)$	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

It is known that the bit permutation can be partitioned into 4 independent bit permutations, mapping the output of 4 Sboxes to the input of 4 Sboxes in the next round.

For convenience, we number the Sboxes in i^{th} round as $Sb_0^i, Sb_1^i, ..., Sb_{s-1}^i$, where $s = n/4$. These Sboxes can be grouped in 2 different ways - the Quotient and Remainder groups, Qx and Rx, defined as

- $Qx = \{Sb_{4x}, Sb_{4x+1}, Sb_{4x+2}, Sb_{4x+3}\}$,
- $Rx = \{Sb_x, Sb_{q+x}, Sb_{2q+x}, Sb_{3q+x}\}$, where $q = \frac{s}{4}, 0 \leq x \leq q-1$.

In PRESENT, $n = 64$ and output bits of $Qx^i = \{Sb^i_{4x}, Sb^i_{4x+1}, Sb^i_{4x+2}, Sb^i_{4x+3}\}$ map to input bits of $Rx^{i+1} = \{Sb^{i+1}_x, Sb^{i+1}_{4+x}, Sb^{i+1}_{8+x}, Sb^{i+1}_{12+x}\}$, this group mapping is defined in Table 7, where the entry (l, m) at row rw and column cl denotes that the l^{th} output bit of the Sbox corresponding to the row rw at i^{th} round will map to the m^{th} input bit of the Sbox corresponding to the column cl at $(i+1)^{\text{th}}$ round. For example, suppose $x = 2$, row and column start at 0, then the entry $(3, 2)$ at row 2 and column 3 means that the 3^{rd} output bit of Sb^i_{10} maps to 2^{nd} input bit of Sb^{i+1}_{14}, thus $P(43) = 58$ (see Table 6).

Table 7. PRESENT group mapping from Qx^i to Rx^{i+1}.

Qx^i \ Rx^{i+1}	Sb^{i+1}_x	Sb^{i+1}_{4+x}	Sb^{i+1}_{8+x}	Sb^{i+1}_{12+x}
Sb^i_{4x}	$(0,0)$	$(1,0)$	$(2,0)$	$(3,0)$
Sb^i_{4x+1}	$(0,1)$	$(1,1)$	$(2,1)$	$(3,1)$
Sb^i_{4x+2}	$(0,2)$	$(1,2)$	$(2,2)$	$(3,2)$
Sb^i_{4x+3}	$(0,3)$	$(1,3)$	$(2,3)$	$(3,3)$

PRESENT bit permutation can be realised in hardware with wires only (no logic gates required). Further, full diffusion is achieved in 3 rounds; from 1 bit to 4, then 4 to 16 and then 16 to 64. But, if there exists Hamming weight 1 to Hamming weight 1 differential transition, or $1-1$ bit differential transition, then there exists consecutive single active bit transitions.

We define $1-1$ bit DDT as a sub-table of the DDT containing Hamming weight 1 differences. Consider some Sbox with the following $1-1$ bit DDT (see Table 8). $\Delta\mathbf{x}$ and $\Delta\mathbf{y}$ denote the differential in the input and output of Sbox respectively. It is evident that this Sbox has differential branch number 2.

It is trivial to see that there exists a single active bit path which results in a differential characteristic with single active Sboxes in each round. Let the input differences be at 3^{rd} bit of $Sb^{(i)}_{15}$. According to $1-1$ bit DDT (Table 8), there exists a transition from 1000 to 1000. From the group mapping (Table 7), 3^{rd} output bit of $Sb^{(i)}_{15}$ maps to 3^{rd} input bit of $Sb^{(i+1)}_{15}$. And then the differential continues from 3^{rd} output bit of $Sb^{(i+1)}_{15}$ to 3^{rd} input bit of $Sb^{(i+2)}_{15}$ and so on. Not only that, if there exists any $1-1$ bit transition (not necessarily $1000 \rightarrow 1000$), one can verify that there always exists some differential characteristic with single active Sbox per round for at least 4 consecutive rounds.

To overcome this problem, we propose a new construction paradigm, "Bad Output must go to Good Input" or BOGI in short. We explain this in the context of the differential of an Sbox, but the analysis is same for linear case also.

Bad Output Must Go to Good Input (BOGI). The existence of the single active bit path is because the bit permutation allows $1-1$ bit transition from some Sbox in i^{th} round to propagate to some Sbox in $(i+1)^{\text{th}}$ round that

Table 8. $1-1$ bit DDT Example 1

$\Delta\mathbf{x}$ \ $\Delta\mathbf{y}$	1000	0100	0010	0001
1000	2	0	0	0
0100	0	0	0	0
0010	0	0	0	0
0001	0	0	0	0

Table 9. $1-1$ bit DDT Example 2

$\Delta\mathbf{x}$ \ $\Delta\mathbf{y}$	1000	0100	0010	0001
1000	0	2	2	0
0100	0	0	0	0
0010	0	0	0	0
0001	0	2	2	0

again would produce $1-1$ bit transition. To overcome such problem, it must be ensured that such path does not exist. In $1-1$ bit DDT, let us define $\Delta\mathbf{x} = \mathtt{x}_3\mathtt{x}_2\mathtt{x}_1\mathtt{x}_0$ be a good input if the corresponding row has all zero entries, else a bad input. Similarly, we define $\Delta\mathbf{y} = \mathtt{y}_3\mathtt{y}_2\mathtt{y}_1\mathtt{y}_0$ be a good output if the corresponding column has all zero entries, else a bad output. In Table 8, 1000 is both bad input and bad output, rest are good.

Consider another $1-1$ bit DDT in Table 9. Let GI, GO, BI, BO denote the set of good inputs, good outputs, bad inputs and bad outputs respectively. Then, in Table 9, $GI = \{0100, 0010\}$, $GO = \{1000, 0001\}$, $BI = \{1000, 0001\}$ and $BO = \{0100, 0010\}$. Or, if we represent these binary strings by integers considering the position of the "1" (rightmost position is 0) in these strings, we may rewrite $GI = \{2, 1\}$, $GO = \{3, 0\}$, $BI = \{3, 0\}$ and $BO = \{2, 1\}$.

An output belonging to BO (bad ouput) could potentially come from a single bit transition through some Sbox in this round. Thus we want to map this active output bit to some GI (good input) in the next round, which guaranteed that it will not propagate to another $1-1$ bit transition. As a result, it avoids single active bit path in 2 consecutive rounds.

BOGI: Let $|BO| \leq |GI|$ and $\pi_1 : BO \to GI$ be an injective map. To ensure that π_1 is an injective map, it is required that $|BO| \leq |GI|$ (the cardinality of the set BO must be less than or equal to the cardinality of the set GI). Let $\pi_2 : GO \to \pi_1(BO)^C$ (the complement of $\pi_1(BO)$) be another injective map. The map π_1 ensures that "Bad Output must go to Good Input". A combined map $\pi : BO \cup GO \to BI \cup GI$ is defined as $\pi(e) = \pi_1(e)$ if and only if $e \in BO$, otherwise $\pi(e) = \pi_2(e)$. For example, consider the Table 9. The injective maps $\pi_1 : \{2, 1\} \to \{2, 1\}$ and $\pi_2 : \{3, 0\} \to \{3, 0\}$ both have 2 choices which altogether make 4 choices for the combined map π. An example BOGI mapping would be $\pi(0) = 0, \pi(1) = 1, \pi(2) = 2, \pi(3) = 3$, which happens to be an identity mapping.

Any choice of π may be used to define the bit permutation. We call these πs *differential BOGI permutations* as derived from $1-1$ bit DDT.

Remark: Similar analysis is done for linear case also. Analogous to $1-1$ bit DDT, analysis is done on the basis of $1-1$ bit LAT and BOGI permutations are found for linear case too. We call them *linear BOGI permutations*. We can now choose any common permutation from the set of both differential and linear BOGI permutations.

BOGI Bit Permutation for GIFT. Let $\pi : \{0, 1, 2, 3\} \rightarrow \{0, 1, 2, 3\}$ be a common permutation from the set of both differential and linear BOGI permutations. Table 10 shows the group mapping.

Table 10. BOGI Bit Permutation mapping from Qx^i to Rx^{i+1}.

Qx^i \ Rx^{i+1}	Sb_x^{i+1}	Sb_{q+x}^{i+1}	Sb_{2q+x}^{i+1}	Sb_{3q+x}^{i+1}
Sb_{4x}^i	$(0, \pi(0))$	$(1, \pi(1))$	$(2, \pi(2))$	$(3, \pi(3))$
Sb_{4x+1}^i	$(1, \pi(1))$	$(2, \pi(2))$	$(3, \pi(3))$	$(0, \pi(0))$
Sb_{4x+2}^i	$(2, \pi(2))$	$(3, \pi(3))$	$(0, \pi(0))$	$(1, \pi(1))$
Sb_{4x+3}^i	$(3, \pi(3))$	$(0, \pi(0))$	$(1, \pi(1))$	$(2, \pi(2))$

Note that we made some left rotations to the rows of the bit mapping, this is because we need the inputs to each Sbox in $(i+1)^{\text{th}}$ round to be coming from 4 different bit positions.

In GIFT, we chose an Sbox that has a common BOGI permutation that is an identity mapping, that is $\pi(i) = i$. Figure 2 illustrates the group mapping from $Q0$ to $R0$ in GIFT-64. The same BOGI permutation is applied to all the q group mappings to form the final n-bit permutation for both version of GIFT.

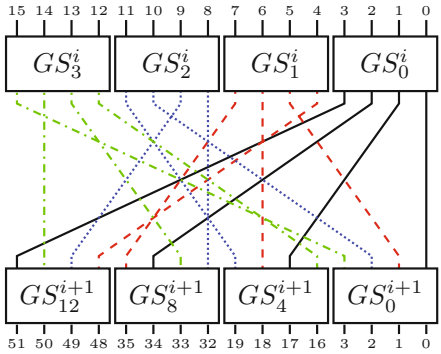

Fig. 2. Group mapping from $Q0$ to $R0$ in GIFT-64.

Some Results About Our Bit Permutation. To be concise, we leave the proofs for our results in the full version. Let $Q0, Q1, \cdots, Q(q-1)$ be q different Quotient groups and $R0, R1, \cdots, R(q-1)$ be q different Remainder groups. Then, for $0 \leq x \leq q-1$,

1. The input bits of an Sbox in Rx come from 4 distinct Sboxes in Qx.
2. The output bits of an Sbox in Qx go to 4 distinct Sboxes in Rx.
3. The input bits of 4 Sboxes from the same Qx come from 16 different Sboxes.
4. The output bits of 4 Sboxes from the same Rx go to 16 different Sboxes.

Lemma 1. *When the number of Sboxes in a round is* 16 *or* 32*, the proposed bit permutation achieves an optimal full diffusion which is achievable by a bit permutation.*

Lemma 2. *In the proposed bit permutation, there does not exist any single active bit transition for two consecutive rounds in both differential and linear characteristics.*

Definition 2. *The* ***differential*** *(resp.* ***linear****)* ***score of an Sbox*** *is* $|GI|+|GO|$ *observed from* $1-1$ *bit DDT (resp. LAT).*

Lemma 3. *There exists differential (resp. linear) BOGI permutation for an Sbox if and only if the differential (resp. linear) score of an Sbox is at least* 4*.*

It is essential that our Sbox has at least score 4 for both differential and linear, and has some common BOGI permutation. These are 2 of the main criteria for the selection of GIFT Sbox.

Remark: BOGI permutation is a group mapping that is independent of the number of groups. Thus, this permutation design is scalable to any bit permutation size that is multiple of 16. This allows us to potentially design larger state size like 256-bit that is useful for designing hash functions.

3.3 Selection of GIFT Sbox

We first recall some Sbox properties and introduce a metric to estimate the hardware implementation cost of Sboxes.

Properties of Sbox. For the differential property, let $S : \mathbb{F}_2^4 \rightarrow \mathbb{F}_2^4$ denote a 4-bit Sbox. Let $\Delta_I, \Delta_O \in \mathbb{F}_2^4$ be the input and output differences, $D_S(\Delta_I, \Delta_O) = \sharp\{x \in \mathbb{F}_2^4 | S(x) \oplus S(x \oplus \Delta_I) = \Delta_O\}$, and $D_{max}(S) = \max_{\Delta_I, \Delta_O \neq 0} D_S(\Delta_I, \Delta_O)$. For the linear property, let $\alpha, \beta \in \mathbb{F}_2^4$ be the input and output masking, $L_S(\alpha, \beta) = |\sharp\{x \in \mathbb{F}_2^4 | x \bullet \alpha = S(x) \bullet \beta\} - 8|$, and $L_{max}(S) = \max_{\alpha,\beta \neq 0} L_S(\alpha, \beta)$.

Definition 3 ([36]). *Let* M_i *and* M_o *be two invertible matrices and* $c_i, c_o \in \mathbb{F}_2^4$*. The Sbox* S' *defined by* $S'(x) = M_o S(M_i(x \oplus c_i)) \oplus c_o$ *belongs to the affine equivalence (AE) set of* S*.*

It is known that both D_{max} and L_{max} are preserved under the AE class.

Definition 4 ([36]). *Let* P_i *and* P_o *be two bit permutation matrices and* $c_i, c_o \in \mathbb{F}_2^4$*. The Sbox* S' *defined by* $S'(x) = P_o S(P_i(x \oplus c_i)) \oplus c_o$ *belongs to the permutation-xor equivalence (PE) set of* S*.*

One is to note that the $1-1$ bit differential and linear transition is preserved only under the PE class. That is to say that the score of an Sbox is preserved under the PE class but not the AE class.

Heuristic Sbox Implementation. We use a simplified metric to estimate the implementation cost of Sboxes. We denote $\{\texttt{NOT}, \texttt{NAND}, \texttt{NOR}\}$ as N-operations[1] and $\{\texttt{XOR}, \texttt{XNOR}\}$ as X-operations, and estimate the cost of an N-operation to be 1 unit and X-operations to be 2 units. We consider the following 4 types of instruction for the construction of the Sboxes: $a \leftarrow \texttt{NOT}(a)$; $a \leftarrow a \texttt{ X } b$; $a \leftarrow a \texttt{ X } (b \texttt{ N } c)$; $a \leftarrow a \texttt{ X } ((b \texttt{ N } c) \texttt{ N } d)$, where a, b, c, d are distinct bits of an Sbox input. These so-called *invertible instructions* [23] allow us to implement the inverse Sbox by simply reversing the sequence of the instructions. In addition, the implementation cost of the inverse Sbox would be the same as the direct Sbox since the same set of instructions is used.

Under this metric, we found that PRESENT Sbox requires $4\texttt{N} + 9\texttt{X}$ operations, a cost of 22 units. While RECTANGLE Sbox requires $4\texttt{N} + 7\texttt{X}$ operations, a cost of 18 units. Hence, one of the criteria for our Sbox is to have implementation cost lesser than 18 units[2].

Search for GIFT Sbox. Our primary design criteria for the GIFT Sbox are:

1. Implementation cost of at most 17 units.
2. With a score of at least 4 in both differential and linear. I.e. For both differential and linear, $|GO| + |GI| \geq 4$.
3. There exists a common BOGI permutation for both differential and linear.

From the list of 302 AE Sboxes presented in [14], we generate the PE Sboxes and check its implementation cost. Our heuristic search shows that there is no optimal Sboxes [30] ($D_{max} = 4$ and $L_{max} = 4$) that satisfies all 3 criteria, hence we extended our search to non-optimal Sboxes. For Sboxes with $D_{max} = 6$ and $L_{max} = 4$, we found some Sboxes with implementation cost of 16 units. For a cost of 15 units, the best possible Sboxes (in terms of D_{max} and L_{max}) that satisfies the criteria have $D_{max} = 12$ and $L_{max} = 6$. And Sboxes with cost of at most 14 units have either $D_{max} = 16$ or $L_{max} = 8$. To maximise the resistance against differential and linear attacks while satisfying the Sbox criteria, we consider Sboxes with $D_{max} = 6$, $L_{max} = 4$ and implementation cost of 16 units.

In order to reduce the occurrence of sub-optimal differential transition, we impose two additional criteria:

4. $\sharp\{(\Delta_I, \Delta_O) \in \mathbb{F}_2^4 \times \mathbb{F}_2^4 | D_S(\Delta_I, \Delta_O) > 4\} \leq 2$.
5. For $D_S(\Delta_I, \Delta_O) > 4$, $wt(\Delta_I) + wt(\Delta_O) \geq 4$, where $wt(\cdot)$ is the Hamming weight.

Criteria (5) ensures that when sub-optimal differential transition occurs, there is a total of at least 4 active Sboxes in the previous and next round.

Finally, we pick an Sbox with a common BOGI permutation for differential and linear that is an identity, i.e. $\pi(i) = i$.

[1] We do not need to consider AND and OR because when we use these invertible instructions, it is equivalent to some other instructions that have been taken into consideration. For instance, a XOR $(b$ AND $c) \equiv a$ XNOR $(b$ NAND $c)$.

[2] This "unit" metric is to facilitate the Sbox search, the Sboxes are later synthesized to obtain their GE in Sect. 5.

Properties of GIFT Sbox. Our GIFT Sbox GS can be implemented with 4N+6X operations (smaller than the Sboxes in PRESENT and RECTANGLE), has a maximum differential probability of $2^{-1.415}$ and linear bias of 2^{-2}, algebraic degree 3 and no fixed point. For the sub-optimal differential transitions with probability $2^{-1.415}$, there are only 2 such transitions and the sum of Hamming weight of input and output differentials is 4. The implementation, differential distribution table (DDT) and linear approximation table (LAT) of GS are provided in the full version.

3.4 Designing of GIFT Key Schedule

Key State Update. One of our main goals when designing the key schedule is to minimize the hardware area, and thus we chose bit permutation which is just wire shuffle and has no hardware area at all. For it to be also software friendly, we consider the entire key state rotation to be in blocks of 16-bit, and bit rotations within some 16-bit blocks. Since it is redundant to apply bit rotations within key state blocks that have not been introduced to the cipher state, we update the key state blocks only after it has been extracted as a round key.

To introduce the entire key material into the cipher state as fast as possible, the key state blocks that are extracted as the round key are chosen such that all the key material are introduced into the cipher state in the least possible number of rounds.

Adding Round Keys. To optimize the hardware performances of GIFT, we XOR the round key to only half of the cipher state. This saves a significant amount of hardware area in a round-based implementation. For it to be software friendly too, we XOR the round key at the same i-th bit positions of each nibble. This makes the bitslice implementation more efficient. In addition, since all nibbles contains some key material, the entire state will be dependent on the key after a SubCells operation.

The choice of the positions for adding the round key and 16-bit rotations were chosen to optimize the related-key differential bounds. However, we would like to reiterate that more rounds is advised to resist related-key attacks.

Round Constants. For the round constants, but instead of using a typical decimal counter, we use a 6-bit affine LFSR (like in SKINNY [5]). It requires only a single XNOR gate per update which is probably has smallest possible hardware area for a counter. Each of the 6 bits is xored to a different nibble to break the symmetry. In addition, we add a "1" at the MSB to further increase the effect.

4 Security Analysis

In this section, we provide short summary of the various cryptanalysis that we had conducted on GIFT. All details are provided in the full version.

4.1 Differential and Linear Cryptanalysis

We use Mixed Integer Linear Programming(MILP) to compute the lower bounds for the number of active Sboxes in both differential cryptanalysis [9] (DC) and linear cryptanalysis [31] (LC), the results are summaries in Table 11. The MILP solution provide us the actual differential or linear characteristics, which allow us to compute the actual differential probability and correlation contribution.

Table 11. Lower bounds for number of active Sboxes.

Cipher	DC/LC	Rounds								
		1	2	3	4	5	6	7	8	9
GIFT-64	DC	1	2	3	5	7	10	13	16	18
	LC	1	2	3	5	7	9	12	15	18
PRESENT	DC	1	2	4	6	10	12	14	16	18
	LC	1	2	3	4	5	6	7	8	9
RECTANGLE	DC	1	2	3	4	6	8	11	13	14
	LC	1	2	3	4	6	8	10	12	14
GIFT-128	DC	1	2	3	5	7	10	13	17	19
	LC	1	2	3	5	7	9	12	14	18

Recall that one of our main goals is to match the differential bounds of PRESENT, that is having an average of 2 active Sboxes per round, but with a lighter Sbox and without the constraint of differential branching number of 3. In addition, we aim for same ratio for the linear bound which was not accomplished by PRESENT. These targets were achieved at 9-round of GIFT. Hence, our DC and LC analysis and discussion focus on 9-round.

Regarding the security against DC, GIFT-64 has a 9-round differential probability of $2^{-44.415}$, taking the average per round and propagate forward, we expect that the differential probability will be lower than 2^{-63} after 14 rounds. Therefore, we believe 28-round GIFT-64 is enough to resist against DC. For GIFT-128, it has a 9-round differential probability of $2^{-46.99}$, which suggested that 26-round is sufficient to achieve a differential probability lower than 2^{-127}. Therefore, we believe 40-round GIFT-128 is enough to resist against DC.

Regarding LC, GIFT-64 has a 9-round linear hull effect of $2^{-49.997}$, which expected to require 13-round to achieve correlation potential lower than 2^{-64}. Therefore, we believe 28-round GIFT-64 is enough to resist against LC. For GIFT-128, it has a 9-round differential probability of $2^{-45.99}$, which means that we would need around 27 rounds to achieve a differential probability lower than 2^{-128}. Therefore, we believe 40-round GIFT-128 is enough to resist against LC.

Related-Key Differential Cryptanalysis. For GIFT-64, since it takes 4 rounds for the all the key material to be introduced into the cipher state, it is trivial to see that it is possible to have no active Sboxes from 1-round to 4-round. Thus we start our computation on the related-key differential bounds

from 5-round onwards. From 5-round to 12-round, the probability of these differential characteristics are $2^{-1.415}, 2^{-5}, 2^{-6.415}, 2^{-10}, 2^{-16}, 2^{-22}, 2^{-27}, 2^{-33}$ respectively. Even if we suppose that the probability of 12-round characteristic is lower bounded by 2^{-33}, it is doubtful that 28 rounds are secure against related-key differential cryptanalysis. Therefore, as we describe in Sect. 2, we strongly recommend to increase the number of rounds to achieve the security against the related-key attacks.

For GIFT-128, we start our computation from 3-round onwards. From 3-round to 9-round, the probabilities are $2^{-1.415}, 2^{-5}, 2^{-7}, 2^{-11}, 2^{-20}, 2^{-25}, 2^{-31}$ respectively. Similar to GIFT-64, it is doubtful that 40 rounds are secure against related-key differential cryptanalysis.

4.2 Integral Attacks

We discuss the security against integral attacks [26]. Here the integral distinguisher is found by using the (bit-based) division property [40,42] and the key recovery is executed by using the partial-sum technique [19]. As a result, the number of rounds that we can find integral distinguishers is 9 rounds for GIFT-64, and the following is an example.

$$(A^{60}, ACAA) \xrightarrow{9R} ((UUBB)^{16})$$

Here, only 2nd bit in plaintext is constant, and bits $\{b_{4i}\}$ and $\{b_{4i+1}\}$ in 9-round ciphertexts are balanced. Note that there is no whitening key at the beginning. Therefore, we can trivially extend integral distinguishers by one round, and GIFT-64 has 10-round integral distinguishers, respectively. We can append four rounds to the 10-round integral distinguisher as the key recovery and attack 14-round GIFT-64. The attack complexity is about 2^{97} with 2^{63} chosen plaintexts.

We also evaluated the longest integral distinguisher for GIFT-128 by using the (bit-based) division property. As a result, we can find 11-round integral distinguisher. The number of rounds is improved by two rounds than that for GIFT-128. However, the number of bits in round key that is XORed every round increases from 32 bits to 64 bits. Therefore, we expect that GIFT-128 is also secure against integral attacks.

4.3 Impossible Differential Attacks

Impossible differential attacks [8,25] exploits a pair of difference Δ_1 and Δ_2 in which Δ_1 never reaches Δ_2 after some rounds.

We searched for impossible differentials by using the MILP-based tool [38]. The results show that there does not exist any impossible differentials with 1-active nibble against 7 rounds of GIFT-64. Thus full rounds are sufficient to resist the impossible differential attack.

4.4 Meet-in-the-Middle Attacks

The meet-in-the-middle (MITM) attack discussed here is a rather classical one, which separates the encryption algorithm into two independent functions [13,16].

GIFT-64-128 XORs only 32 bits out of 128 bits of the key to the state in every round. Given this property, along with splice-and-cut [1] and initial-structure (IS) [37] techniques, we choose that 8 bits of (k_6, k_7) and 8 bits of k_2, k_3 as sources of independent computations called neutral bits and separate 15 rounds as shown in Fig. 3. Note that when the backward computation reaches the plaintext, the attacker makes a query to obtain the corresponding ciphertext. Every details of the attack procedure will be explained in the full version.

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Subkey U	k_1	k_3^B	k_5	k_7^F	k_1	k_3^B	k_5	k_7^F	k_1	k_3^B	k_5	k_7^F	k_1	k_3^B	k_5
V	k_0	k_2^B	k_4	k_6^F	k_0	k_2^B	k_4	k_6^F	k_0	k_2^B	k_4	k_6^F	k_0	k_2^B	k_4
Remarks	$\longleftarrow$			IS			$\longrightarrow$			match			$\longleftarrow$		

Fig. 3. Chunk separation for 15-round MitM attack.

For each of 2^{112} non-neutral bits, the attacker computes the forward and backward chunks for 2^8 choices of neutral bits. Therefore, the time complexity is 2^{120} and the memory complexity is 2^8. This requires the knowledge of the full codebook, thus the data complexity is 2^{64}.

4.5 Invariant Subspace Attacks

Since the round constant is XORed only in the MSB of several S-boxes, invariant subspace attacks [20,28,29] can be a potential threat.

We exhaustively searched for the subspace transition through the GIFT S-box and confirmed that XORing the constant to MSB breaks the invariant subspace, thus GIFT resists the attack. The details are provided in the full version.

4.6 Nonlinear Invariant Attacks

Nonlinear invariant attacks [41] are weak-key attacks that can be applied when the round constant is XORed only to some particular bits of nibbles. The core idea is to find a nonlinear approximation of the round transformation with probability one. For the SPN structure, the attacks are mounted when (1) S-box has the quadratic nonlinear invariant and (2) the linear layer is represented by the multiplication with an orthogonal binary matrix.

The diffusion of GIFT (bit permutation) is orthogonal. However, it is not represented by the multiplication with an orthogonal binary matrix. Moreover, we searched for the quadratic nonlinear invariant for GIFT S-box, but there is no such invariant. Therefore, GIFT is secure against the nonlinear invariant attacks.

4.7 Algebraic Attacks

Algebraic attacks do not threaten GIFT, the analysis is provided in the full version.

5 Hardware Implementation

GIFT is surprisingly efficient and on ASIC platforms across various degrees of serialization. This is mainly due to the extremely lightweight round function that performs key addition on only half of the state and uses a bit permutation as the only diffusion mechanism. Due to page constraints, we leave the details in the full version of our paper and present the summary here.

5.1 Round Based Implementation

GIFT includes various design strategies in order to minimize gate count. GIFT employs key addition to only half of the state and so saves silicon area in the process. SKINNY uses the same mechanism, but it additionally uses an equal amount of XOR gates to add the tweak to the state, and so the number of XOR gates required to construct the roundkey addition layer is equal to that of any cipher employing full state addition.

In Table 12, we compare the hardware performances of GIFT with other lightweight ciphers. In Fig. 4 we list the individual area requirements of the respective components in GIFT.

We see that GIFT has the smallest area compared to the other ciphers. From the pie chart, we see that the storage area (which is a fixed cost) took up most of the area percentage, the cipher component (which is the variable) only make up a small percentage to the overall area.

Table 12. Comparison of performance metrics for round based implementations synthesized with STM 90 nm Standard cell library

	Area (GE)	Delay (ns)	Cycles	TP_{MAX} (MBit/s)	Power (μW) (@10 MHz)	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
MIDORI 64	1542	2.06	17	1941.7	60.6	103.0
PICCOLO 64/128[a]	1868	2.32	32	889.9	79.4	254.1
RECTANGLE 64/128	1637	1.61	27	1472.2	76.2	206.0
LED 64/128	1831	5.25	50	243.8	131.3	656.5
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
MIDORI 128	2522	2.25	21	2844.4	89.2	187.3
AES128	7215	3.83	11	3038.2	730.3	803.3

([a] Piccolo implemented in dynamic key mode)

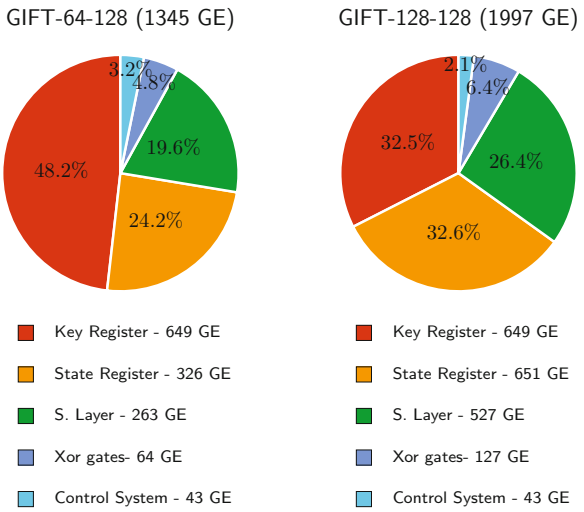

Fig. 4. Componentwise area requirements for `GIFT-64-128` and `GIFT-128-128`

5.2 Serial Implementation

The serial implementation of `GIFT-64-128` uses a mixed datapath of size 4 bits on the stateside and 16 bits on the keyside. The architecture has been explained in Fig. 5.

`GIFT-128-128` uses a similar architecture: a mixture of 4 bit datapath in the stateside and a 32 bit datapath on the keyside is employed. We also implemented bit serial versions of `GIFT` as per the techniques outlined in [24]. In Table 13, we

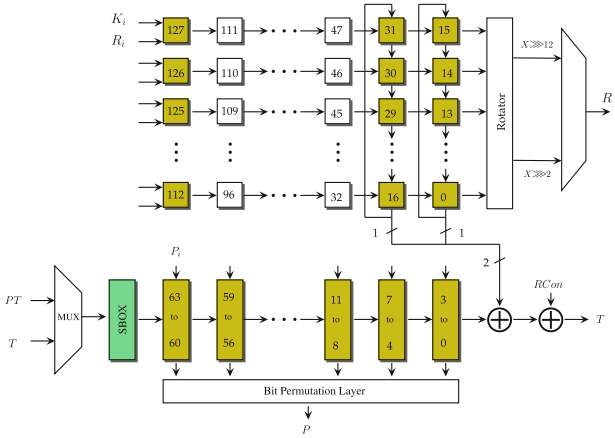

Fig. 5. Serial Implementation for `GIFT-64-128` (The boxes in green denote scan flip-flops/registers)

Table 13. Comparison of performance metrics for serial implementations synthesized with STM 90 nm Standard cell library

	Degree of Serialization	Area (GE)	Delay (ns)	Cycles	TP_{MAX} (MBit/s)	Power (μW) (@10 MHz)	Energy (nJ)
GIFT-64-128	4/16	1113	2.14	522	57.3	39.0	2.04
GIFT-64-128	1	930	2.67	2816	8.5	35.9	10.11
SKINNY-64-128	4	1265	1.73	756	48.9	59.2	4.48
SKINNY-64-128	1	887	0.98	3152	20.7	42.6	13.42
PRESENT 64/128	4	1158	1.94	576	57.3	58.0	3.34
SIMON 64/128	1	794	1.10	1536	37.9	44.7	6.87
LED 64/128	4	1225	2.54	1904	13.2	49.8	9.48
GIFT-128-128	4/32	1455	2.25	714	79.7	61.7	4.40
GIFT-128-128	1	1213	2.46	6528	8.0	40.3	26.30
SKINNY-128-128	8	1638	1.95	840	78.1	79.1	6.64
SKINNY-128-128	1	1110	0.81	6976	22.7	53.8	37.53
SIMON 128/128	1	1077	1.17	4480	25.1	60.5	27.10
AES 128[a]	8	2060	5.79	246	88.6	129.7	3.19

([a] AES implementation figures from [3])

list the performance comparisons of GIFT with other block ciphers. While the bit serial implementation of Simon is probably the most compact due to the nature of the design, but the performance of GIFT is comparable/better with other ciphers with similar level of serialization.

6 Software Implementation

In this section, we describe our software implementation of GIFT-64 and GIFT-128. Due to its inherent bitslice structure, it seems natural to consider that the most efficient software implementations of GIFTwill be bitslice implementations.

We leave the details of the packing/unpacking of the data and round function implementation in the full version.

Benchmarks. We have produced this bitslice implementation for AVX2 registers and we give in Table 14 the benchmarking results on a computer with a Intel Haswell processor (i5-4460U). We have benchmarked the bitslice implementations of SIMON and SKINNY (available online) on the same computer for fairness.

Comments. Bitslice implementations can be used for any parallel mode (as it is the case for most modern operating modes), but can also be used for serial modes when several users are communicating in parallel. In this setting, the

Table 14. Bitslice software implementations of GIFT and other lightweight block ciphers. Performances are given in cycles per byte, with messages composed of 2000 64-bit blocks to obtain the results.

Cipher	Speed (c/B)	Ref.	Cipher	Speed (c/B)	Ref.
GIFT-64-128	2.10	**new**	GIFT-128-128	2.57	**new**
SKINNY-64-128	2.88	[27]	SKINNY-128-128	4.70	[27]
SIMON-64-128	1.74	[43]	SIMON-128-128	2.55	[43]

implementation would be exactly the same, as our key preparation does not assume that the keys have to be the same for all blocks. In the scenario of a serial mode for a single user, then a classical table-based or VPERM implementation will probably be the most efficient option [6].

For low-end micro-controllers, it is very likely that GIFT will perform very well on this platform. RECTANGLE is very good on micro-controllers and GIFT shares the same general strategy on this regard. The key schedule being even simpler, we believe that it will actually perform even better than RECTANGLE.

Acknowledgements. The authors would like to thank the anonymous referees for their helpful comments. This work is partly supported by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06).

References

1. Aoki, K., Sasaki, Y.: Preimage attacks on one-block MD4, 63-step MD5 and more. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 103–119. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04159-4_7
2. 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.) ASIACRYPT 2015. LNCS, vol. 9453, pp. 411–436. Springer, Heidelberg (2015). doi:10.1007/978-3-662-48800-3_17
3. Banik, S., Bogdanov, A., Regazzoni, F.: Atomic-AES v 2.0. Cryptology ePrint Archive, Report 2016/1005 (2016)
4. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK Families of Lightweight Block Ciphers. Cryptology ePrint Archive, Report 2013/404 (2013)
5. 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.) CRYPTO 2016. LNCS, vol. 9815, pp. 123–153. Springer, Heidelberg (2016). doi:10.1007/978-3-662-53008-5_5
6. Benadjila, R., Guo, J., Lomné, V., Peyrin, T.: Implementing lightweight block ciphers on x86 architectures. In: Lange, T., Lauter, K., Lisoněk, P. (eds.) SAC 2013. LNCS, vol. 8282, pp. 324–351. Springer, Heidelberg (2014). doi:10.1007/978-3-662-43414-7_17

7. Biham, E., Anderson, R., Knudsen, L.: Serpent: a new block cipher proposal. In: Vaudenay, S. (ed.) FSE 1998. LNCS, vol. 1372, pp. 222–238. Springer, Heidelberg (1998). doi:10.1007/3-540-69710-1_15
8. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. J. Cryptology **18**(4), 291–311 (2005)
9. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. In: Menezes, A.J., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 2–21. Springer, Heidelberg (1991). doi:10.1007/3-540-38424-3_1
10. Blondeau, C., Nyberg, K.: Links between truncated differential and multidimensional linear properties of block ciphers and underlying attack complexities. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 165–182. Springer, Heidelberg (2014). doi:10.1007/978-3-642-55220-5_10
11. Bogdanov, A., Knežević, M., Leander, G., Toz, D., Varıcı, K., Verbauwhede, I.: SPONGENT: a lightweight hash function. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 312–325. Springer, Heidelberg (2011). doi:10.1007/978-3-642-23951-9_21
12. 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.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007). doi:10.1007/978-3-540-74735-2_31
13. Bogdanov, A., Rechberger, C.: A 3-subset meet-in-the-middle attack: cryptanalysis of the lightweight block cipher KTANTAN. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010. LNCS, vol. 6544, pp. 229–240. Springer, Heidelberg (2011). doi:10.1007/978-3-642-19574-7_16
14. Cannière, C.D.: Analysis and Design of Symmetric Encryption Algorithms. Ph.D thesis, Katholieke Universiteit Leuven Bart Preneel (promotor) (2007)
15. Cannière, C., Dunkelman, O., Knežević, M.: KATAN and KTANTAN — a family of small and efficient hardware-oriented block ciphers. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 272–288. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04138-9_20
16. Chaum, D., Evertse, J.-H.: Cryptanalysis of des with a reduced number of rounds. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 192–211. Springer, Heidelberg (1986). doi:10.1007/3-540-39799-X_16
17. Cho, J.Y.: Linear cryptanalysis of reduced-round PRESENT. In: Pieprzyk, J. (ed.) CT-RSA 2010. LNCS, vol. 5985, pp. 302–317. Springer, Heidelberg (2010). doi:10.1007/978-3-642-11925-5_21
18. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer, Heidelberg (2002)
19. Ferguson, N., Kelsey, J., Lucks, S., Schneier, B., Stay, M., Wagner, D., Whiting, D.: Improved cryptanalysis of rijndael. In: Goos, G., Hartmanis, J., Leeuwen, J., Schneier, B. (eds.) FSE 2000. LNCS, vol. 1978, pp. 213–230. Springer, Heidelberg (2001). doi:10.1007/3-540-44706-7_15
20. Guo, J., Jean, J., Nikolic, I., Qiao, K., Sasaki, Y., Sim, S.: Invariant subspace attack against midori64 and the resistance criteria for s-box designs. IACR Trans. Symmetric Cryptology **2016**(1), 33–56 (2016)
21. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON family of lightweight hash functions. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 222–239. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22792-9_13
22. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.: The LED block cipher. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 326–341. Springer, Heidelberg (2011). doi:10.1007/978-3-642-23951-9_22. [35]

23. Jean, J., Peyrin, T., Sim, S.M.: Optimizing implementations of lightweight building blocks. Cryptology ePrint Archive, Report 2017/101 (2017)
24. Jean, J., Moradi, A., Peyrin, T., Sasdrich, P.: Bit-Sliding: A Generic Technique for Bit-Serial Implementations of SPN-based Primitives. In: To appear in Cryptographic Hardware and Embedded Systems - CHES 2017 - Taipei, Taiwan, 25–28 September 2017
25. Knudsen, L.: Deal - a 128-bit block cipher. NIST AES Proposal (1998)
26. Knudsen, L., Wagner, D.: Integral cryptanalysis. In: Daemen, J., Rijmen, V. (eds.) FSE 2002. LNCS, vol. 2365, pp. 112–127. Springer, Heidelberg (2002). doi:10.1007/3-540-45661-9_9
27. Kölbl, S.: AVX implementation of the Skinny block cipher (2016). https://github.com/kste/skinny_avx
28. Leander, G., Abdelraheem, M.A., AlKhzaimi, H., Zenner, E.: A cryptanalysis of PRINTcipher: the invariant subspace attack. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 206–221. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22792-9_12
29. Leander, G., Minaud, B., Rønjom, S.: A generic approach to invariant subspace attacks: cryptanalysis of robin, iSCREAM and zorro. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 254–283. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46800-5_11
30. Leander, G., Poschmann, A.: On the classification of 4 bit S-boxes. In: Carlet, C., Sunar, B. (eds.) WAIFI 2007. LNCS, vol. 4547, pp. 159–176. Springer, Heidelberg (2007). doi:10.1007/978-3-540-73074-3_13
31. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994). doi:10.1007/3-540-48285-7_33
32. Nakahara, J.: 3D: a three-dimensional block cipher. In: Franklin, M.K., Hui, L.C.K., Wong, D.S. (eds.) CANS 2008. LNCS, vol. 5339, pp. 252–267. Springer, Heidelberg (2008). doi:10.1007/978-3-540-89641-8_18
33. National Institute of Standards and Technology: Fips 180–2: Secure hash standard. http://csrc.nist.gov
34. National Institute of Standards and Technology: Lightweight cryptography (2016). https://www.nist.gov/programs-projects/lightweight-cryptography
35. Preneel, B., Takagi, T. (eds.): CHES 2011. LNCS, vol. 6917. Springer, Heidelberg (2011)
36. Saarinen, M.-J.O.: Cryptographic analysis of all 4 × 4-bit S-boxes. In: Miri, A., Vaudenay, S. (eds.) SAC 2011. LNCS, vol. 7118, pp. 118–133. Springer, Heidelberg (2012). doi:10.1007/978-3-642-28496-0_7
37. Sasaki, Y., Aoki, K.: Finding preimages in full MD5 faster than exhaustive search. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 134–152. Springer, Heidelberg (2009). doi:10.1007/978-3-642-01001-9_8
38. Sasaki, Y., Todo, Y.: New impossible differential search tool from design and cryptanalysis aspects. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10212, pp. 185–215. Springer, Cham (2017). doi:10.1007/978-3-319-56617-7_7
39. Shibutani, K., Isobe, T., Hiwatari, H., Mitsuda, A., Akishita, T., Shirai, T.: *Piccolo*: an ultra-lightweight blockcipher. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 342–357. Springer, Heidelberg (2011). doi:10.1007/978-3-642-23951-9_23. [35]
40. Todo, Y.: Structural evaluation by generalized integral property. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 287–314. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46800-5_12

41. Todo, Y., Leander, G., Sasaki, Y.: Nonlinear invariant attack. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10032, pp. 3–33. Springer, Heidelberg (2016). doi:10.1007/978-3-662-53890-6_1
42. Todo, Y., Morii, M.: Bit-based division property and application to Simon family. In: Peyrin, T. (ed.) FSE 2016. LNCS, vol. 9783, pp. 357–377. Springer, Heidelberg (2016). doi:10.1007/978-3-662-52993-5_18
43. Wingers, L.: Software for SUPERCOP benchmarking of SIMON and SPECK (2015). https://github.com/lrwinge/simon_speck_supercop
44. Zhang, W., Bao, Z., Lin, D., Rijmen, V., Yang, B., Verbauwhede, I.: Rectangle: a bit-slice lightweight block cipher suitable for multiple platforms. Sci. China Inf. Sci. **58**(12), 1–15 (2015)