Fixslicing: A New GIFT Representation

Fast Constant-Time Implementations of GIFT and GIFT-COFB on ARM Cortex-M

Alexandre Adomnicai[1,2] Zakaria Najm[1,2,3] Thomas Peyrin[1,2]

[1]Nanyang Technological University, Singapore

[2]Temasek Laboratories, Singapore

[3]TU Delft, The Netherlands

CHES 2020: International Conference on Cryptographic Hardware and Embedded Systems

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

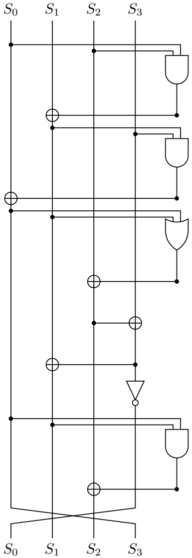
Some context

- **Lightweight crypto** has been a very hot topic in the past decade
- **100+ ciphers** claiming to be *lightweight* have been published in the literature
- No single algorithm is more efficient than all others on **every possible platforms**
- Designs are usually **hardware or software oriented**
- **How efficient hardware-oriented ciphers can be in software?**
- Important question for the ongoing **NIST LWC standardization project**

The GIFT family of block ciphers

- Introduced at CHES 2017 with 2 different block sizes: GIFT-64 and GIFT-128
- GIFT block ciphers are Substitution-bitPermutation Networks (SbPN) i.e. **the linear layer only consists of a bit permutation** ⇒ **hardware-oriented** design
- Improvement of the 64-bit cipher PRESENT (ISO/IEC 29192 standard)
  - **Smaller area** thanks to a smaller S-box and lesser subkey additions
  - **Better resistance against linear cryptanalysis** thanks to its building blocks' properties
  - **Higher throughput**
  - Extend to **128-bit block size**
- Used in several **NIST LWC round 2 candidates**: GIFT-COFB, SUNDAE-GIFT, HYENA, ESTATE, LOTUS/LOCUS

4-bit S-box



$$
\begin{aligned}
S_1 &\leftarrow S_1 \oplus (S_0 \wedge S_2) \\
S_0 &\leftarrow S_0 \oplus (S_1 \wedge S_3) \\
S_2 &\leftarrow S_2 \oplus (S_0 \vee S_1) \\
S_3 &\leftarrow S_3 \oplus S_2 \\
S_1 &\leftarrow S_1 \oplus S_3 \\
S_3 &\leftarrow \neg S_3 \\
S_2 &\leftarrow S_2 \oplus (S_0 \wedge S_1) \\
\{S_0, S_1, S_2, S_3\} &\leftarrow \{S_3, S_1, S_2, S_0\},
\end{aligned}
$$

- ▷ algebraic degree 3
- ▷ 12 instructions in total (4 non-linear)

Bit permutation used in GIFT-64

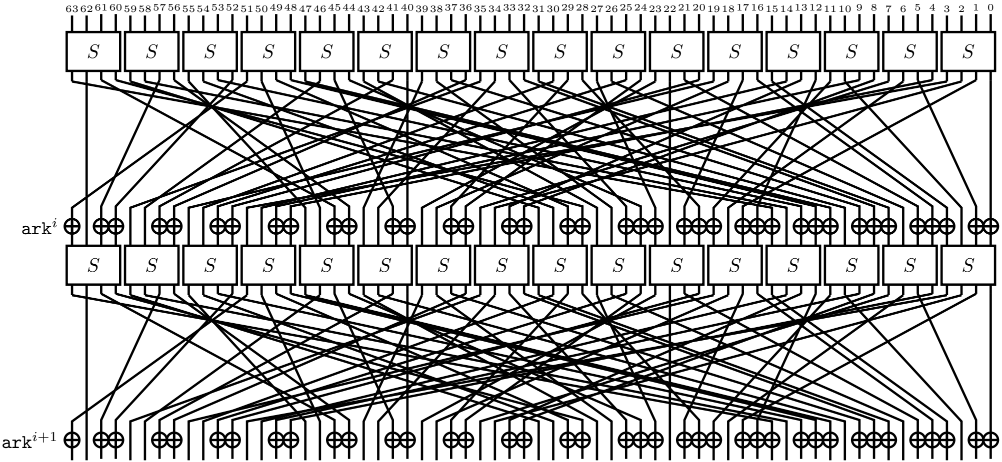


Figure: 2 rounds of GIFT-64 (from https://www.iacr.org/authors/tikz/)

Bit permutation used in GIFT-64

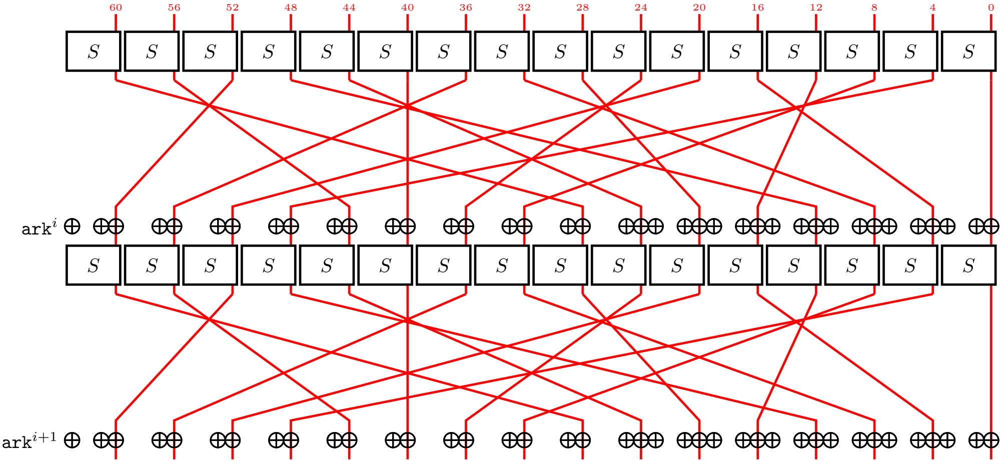


Figure: 2 rounds of GIFT-64 (from https://www.iacr.org/authors/tikz/)

Bit permutation used in GIFT-64

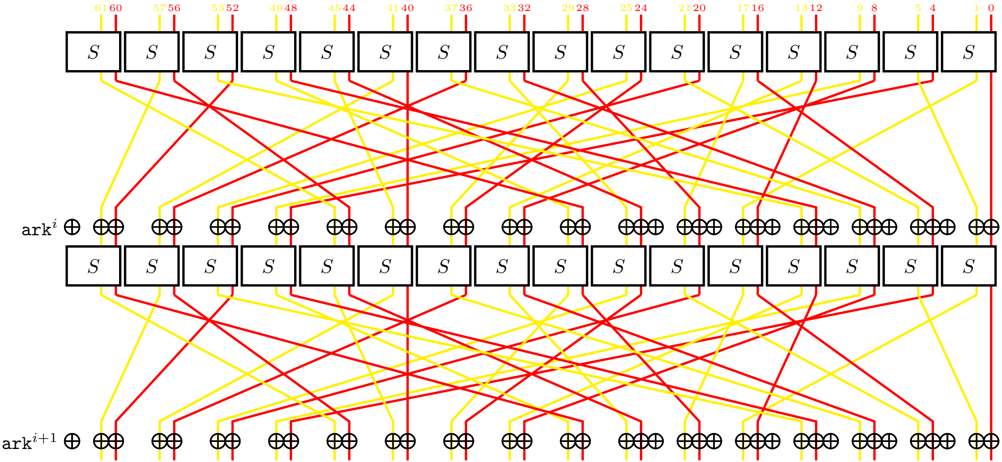


Figure: 2 rounds of GIFT-64 (from https://www.iacr.org/authors/tikz/)

Bit permutation used in GIFT-64

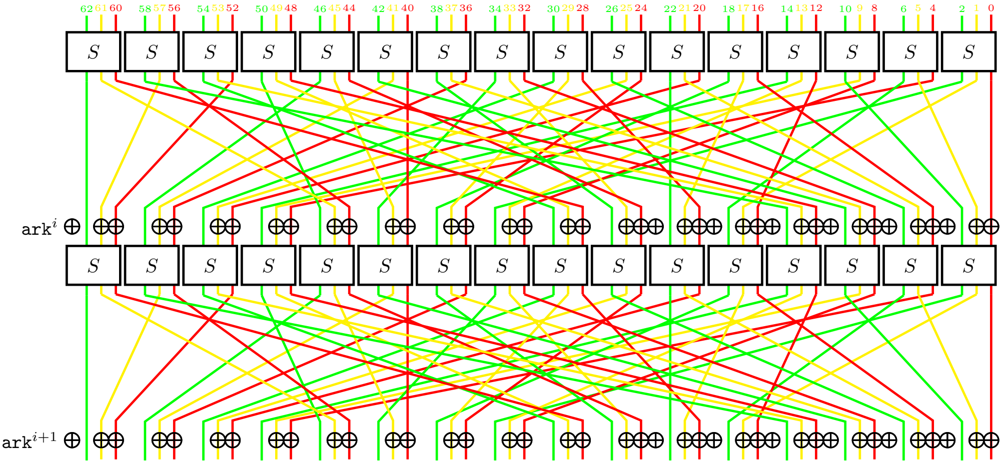


Figure: 2 rounds of GIFT-64 (from https://www.iacr.org/authors/tikz/)

Bit permutation used in GIFT-64

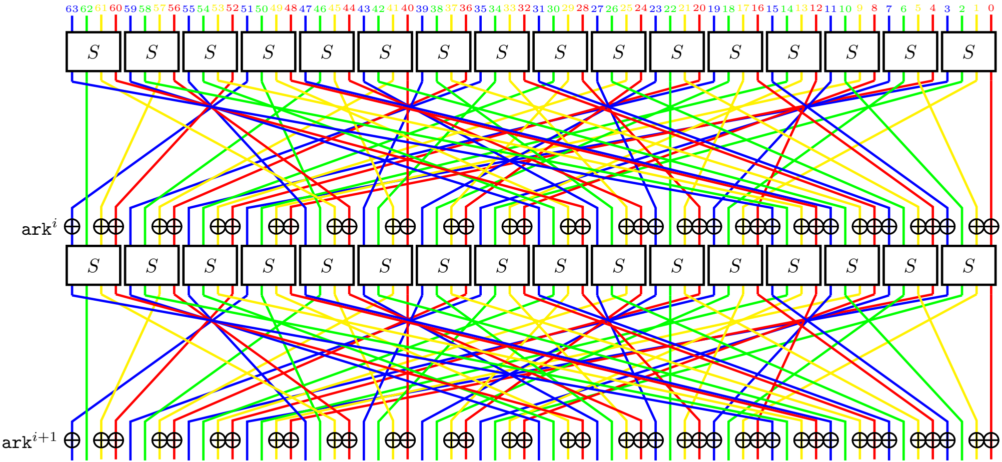


Figure: 2 rounds of GIFT-64 (from https://www.iacr.org/authors/tikz/)

Bit permutation used in GIFT-64: software implementation

$$S = \begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{bmatrix} \leftarrow \begin{bmatrix} b_{60} & \cdots & b_8 & b_4 & b_0 \\ b_{61} & \cdots & b_9 & b_5 & b_1 \\ b_{62} & \cdots & b_{10} & b_6 & b_2 \\ b_{63} & \cdots & b_{11} & b_7 & b_3 \end{bmatrix}$$

▷ **Each bit located in a slice remains in the same slice through the bit permutation** $\Rightarrow$ different permutations are applied to each S_i independently

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P_0(j)$	0	12	8	4	1	13	9	5	2	14	10	6	3	15	11	7
$P_1(j)$	4	0	12	8	5	1	13	9	6	2	14	10	7	3	15	11
$P_2(j)$	8	4	0	12	9	5	1	13	10	6	2	14	11	7	3	15
$P_3(j)$	12	8	4	0	13	9	5	1	14	10	6	2	15	11	7	3

Bit permutation used in GIFT-64: software implementation

$$
\begin{aligned}
P_0(S_0) = (S_0 \wedge \texttt{0x0401}) \quad & \vee \quad ((S_0 \wedge \texttt{0x0008}) \ll 1) \quad \vee \\
((S_0 \wedge \texttt{0x2000}) \ll 2) \quad & \vee \quad ((S_0 \wedge \texttt{0x0040}) \ll 3) \quad \vee \\
((S_0 \wedge \texttt{0x0200}) \ll 5) \quad & \vee \quad ((S_0 \wedge \texttt{0x0004}) \ll 6) \quad \vee \\
((S_0 \wedge \texttt{0x0020}) \ll 8) \quad & \vee \quad ((S_0 \wedge \texttt{0x0002}) \ll 11) \quad \vee \\
((S_0 \wedge \texttt{0x1000}) \gg 9) \quad & \vee \quad ((S_0 \wedge \texttt{0x8000}) \gg 8) \quad \vee \\
((S_0 \wedge \texttt{0x0100}) \gg 6) \quad & \vee \quad ((S_0 \wedge \texttt{0x0800}) \gg 5) \quad \vee \\
((S_0 \wedge \texttt{0x4010}) \gg 3) \quad & \vee \quad ((S_0 \wedge \texttt{0x0080}) \gg 2)
\end{aligned}
$$

- ▷ The entire linear layer requires about 100 cycles per round on ARM Cortex-M processors
- ▷ Possibility to process 2 blocks in parallel on 32-bit platforms to mitigate costs

Naive bitsliced implementation results

Algorithm	Parallel Blocks	Speed (cycles/block)		ROM (bytes)		RAM (bytes)	
		M3	M4	Code	Data	I/O	Stack
GIFT-64	2	2 141	2 138	1 608	28	52	48
GIFT-128	1	8 644	8 573	1 996	40	52	48

Table: Constant-time implementation results on ARM Cortex-M3 and M4

- GIFT-64 and GIFT-128 run at **268 and 540 cycles/Byte on ARM Cortex-M3/4**
- AES-128 runs at **101 cycles/Byte on the same platform** by processing 2 blocks in parallel [SS16]

Bitsliced representation of GIFT-64 (over 4 rounds)

slice 0

0	4	8	12
16	20	24	28
32	36	40	44
48	52	56	60

slice 1

1	5	9	13
17	21	25	29
33	37	41	45
49	53	57	61

slice 2

2	6	10	14
18	22	26	30
34	38	42	46
50	54	58	62

slice 3

3	7	11	15
19	23	27	31
35	39	43	47
51	55	59	63

Bitsliced representation of GIFT-64 (over 4 rounds)

slice 0

0	4	8	12
16	20	24	28
32	36	40	44
48	52	56	60

slice 1

1	5	9	13
17	21	25	29
33	37	41	45
49	53	57	61

slice 2

2	6	10	14
18	22	26	30
34	38	42	46
50	54	58	62

slice 3

3	7	11	15
19	23	27	31
35	39	43	47
51	55	59	63

0	16	32	48
12	28	44	60
8	24	40	56
4	20	36	52

5	21	37	53
1	17	33	49
13	29	45	61
9	25	41	57

10	26	42	58
6	22	38	54
2	18	34	50
14	30	46	62

15	31	47	63
11	27	43	59
7	23	39	55
3	19	35	51

Bitsliced representation of GIFT-64 (over 4 rounds)

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

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

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

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Bitsliced representation of GIFT-64 (over 4 rounds)

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

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

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

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

Bitsliced representation of GIFT-64 (over 4 rounds)

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

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

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

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

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

Some properties on the bit permutation used in GIFT-64

- $P_i^4 = Id$ for all $i \Rightarrow$ **all bits are back at their original position every 4 rounds**

- Following an alternative representation for a few rounds might help

- A decomposition of the PRESENT permutation over 2 rounds allows significant performance improvements [RAL17]

- **What if we completely omit the permutation for a given slice?**

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Fixsliced representation of GIFT-64 (over 4 rounds)

slice 0

0	4	8	12
16	20	24	28
32	36	40	44
48	52	56	60

slice 1

1	5	9	13
17	21	25	29
33	37	41	45
49	53	57	61

slice 2

2	6	10	14
18	22	26	30
34	38	42	46
50	54	58	62

slice 3

3	7	11	15
19	23	27	31
35	39	43	47
51	55	59	63

Fixsliced representation of GIFT-64 (over 4 rounds)

slice 0

0	4	8	12
16	20	24	28
32	36	40	44
48	52	56	60

0	4	8	12
16	20	24	28
32	36	40	44
48	52	56	60

slice 1

1	5	9	13
17	21	25	29
33	37	41	45
49	53	57	61

←

5	9	13	1
21	25	29	17
37	41	45	33
53	57	61	49

slice 2

2	6	10	14
18	22	26	30
34	38	42	46
50	54	58	62

←←

10	14	2	6
26	30	18	22
42	46	34	38
58	62	50	54

slice 3

3	7	11	15
19	23	27	31
35	39	43	47
51	55	59	63

←←←

15	3	7	11
31	19	23	27
47	35	39	43
63	51	55	59

Fixsliced representation of GIFT-64 (over 4 rounds)

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

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

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

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Fixsliced representation of GIFT-64 (over 4 rounds)

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

Fixsliced representation of GIFT-64 (over 4 rounds)

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

← ←← ←←←

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

↑ ↑↑ ↑↑↑

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

→ →→ →→→

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

↓ ↓↓ ↓↓↓

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

Fixsliced GIFT-64

- Our new representation consists in fixing a slice to never move and adjust the others accordingly so that the bits are correctly aligned for the S-box $\Rightarrow$ we call it **"fixslicing"**

- For GIFT-64, the slices adjustment consists of **row-wise (↑↓) and column-wise rotations (→←)** depending on the round numbers

- By processing 2 blocks at a time on 32-bit architectures, they can be computed by means of **word-wise and byte-wise rotations**, respectively

- Since word-wise rotations can be computed for free on ARM thanks to the inline barrel shifter, it means that **the linear layer is free every 2 rounds** on those processors

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Application to GIFT-128

- For GIFT-128 we don't have $P_i^4 = Id$ but $P_0^{31} = P_1^{10} = P_2^{31} = P_3^5 = Id$ instead
- **By fixing S_3 to never move we can define an alternative representation that will be synchronized with the classical representation after 5 rounds**
- The slices adjustment are similar to GIFT-64 for the first 2 rounds but are slightly more costly for the last 3 rounds

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Implementation results on ARM Cortex-M

Algorithm	Ref	Parallel Blocks	Speed (cycles/block) M3	Speed (cycles/block) M4	ROM (bytes) Code	ROM (bytes) Data	RAM (bytes) I/O	RAM (bytes) Stack
64-bit ciphers with 128-bit key								
GIFTb-64	Ours	2	383	383	2666	0	40	48
GIFT-64	Ours	2	419	419	2962	0	40	48
PRESENT	[RAL17]	2	1058	800	2476	•	•	•
RECTANGLE	[DCK+19]	1	854	•	800		76	24
SIMON-64	[DCK+19]	1	650	•	456		48	24
SPECK-64	[DCK+19]	1	285	•	628		36	24
128-bit ciphers with 128-bit key								
AES-128	[SS16]	2	1 617	1 618	12 120	12	48	108
GIFTb-128	Ours	1	1 169	1 172	4 250	0	48	56
GIFT-128	Ours	1	1 316	1 319	4 868	0	48	56

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Results interpretation

- Fixslicing allows **significant performance improvement** over naive bitslicing on ARM Cortex-M
  - GIFT-64 runs **5.1x faster**
  - GIFT-128 runs **6.5x faster**
- Our fixsliced representations **perfectly fit the ARM architecture** thanks to the inline barrel shifter
- We expect slightly lower but still impressive improvement factors on **platforms without rotate instructions**

Taking 1st-order masking into consideration

- ▷ Embedded devices are typical targets for **power side-channel attacks** (e.g. DPA)
- ▷ We integrated **1st-order masking** to our fixsliced GIFT implementations

Algorithm	Ref	Parallel Blocks	Speed (cycles per block)	ROM (bytes)		RAM (bytes)	
				Code	Data	I/O	Stack
128-bit ciphers with 128-bit key							
AES-128	[SS16]	2	5 290 (+2133)	39 916	12	48	1588
GIFTb-128	Ours	1	2 815 (+196)	10 266	0	48	64
GIFT-128	Ours	1	2 972 (+196)	10 906	0	48	64

Table: Masked constant-time implementation results on ARM Cortex-M4. For encryption routines, speed is expressed in cycles per block. Number enclosed in parathensis refer to cycles spent for the randomness generation. Implementations are fully unrolled for speed optimization.

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Integration into the GIFT-COFB authenticated cipher

Algorithm	**Ref**	**Speed** (cycles)		**ROM** (bytes)		**RAM** (bytes)	
		M3	M4	Code	Data	I/O	Stack
Without masking							
GIFT-COFB	Ours	4 827	4 893	10 092	0	428	92
Ascon-128	https://github.com/ascon (Our measurements)	4 203	4 276	12 348	0	124	36
Ascon-128a		3 862	3 990	15 200	0	140	36
With 1st-order masking (including randomness generation)							
GIFT-COFB	Ours	•	10 978 (+579)	19 808	0	732	108

Table: Constant-time implementation results on ARM Cortex-M3 and M4 to secure 16 bytes of message along with 16 bytes of additional data. Implementations are fully unrolled for speed optimization.

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Conclusion

- We introduced a **new alternative representation** of GIFT called fixslicing
- Fixslicing allows a **constant-time** and **software-friendly** implementation of the bit permutation
- Fixslicing makes GIFT **extremly efficient in software**, placing GIFT-COFB among the fastest NIST LWC round 2 candidates on microcontrollers
- GIFT is **well suited to side-channel countermeasures** thanks to its S-box properties (only 4 non-linear gates)
- All our implementations are publicly available at `https://github.com/aadomn/gift`

Perspectives

- The fixslicing implementation strategy tends to be **generic**
- **Application to other designs**, not only SbPN structures

Perspectives

- ▷ The fixslicing implementation strategy tends to be **generic**
- ▷ **Application to other designs**, not only SbPN structures

Spoiler Alert!

- ▷ **Fixslicing the** AES **led to new bitsliced speed records** on ARM Cortex-M and RISC-V
  - Will soon appear on eprint
  - Source code available soon at `https://github.com/aadomn/aes`

Thanks for your attention!

Questions?

Feel free to contact us at `firstname.lastname@ntu.edu.sg`

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

References

Daniel Dinu, Yann Le Corre, Dmitry Khovratovich, Léo Perrin, Johann Großschädl, and Alex Biryukov, Triathlon of lightweight block ciphers for the Internet of things, J. Cryptographic Engineering 9 (2019), no. 3, 283–302.

Tiago B. S. Reis, Diego F. Aranha, and Julio López, PRESENT runs fast - efficient and secure implementation in software, Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings, 2017, pp. 644–664.

Peter Schwabe and Ko Stoffelen, All the AES You Need on Cortex-M3 and M4, Selected Areas in Cryptography - SAC 2016, 2016, pp. 180–194.