Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.30, NO.3, Jun. 2020 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2020.30.3.349

효율적인 비트 슬라이스 구현이 가능한 GIFT-64-variant 개발 및 안전성 분석*

백 승 준,[1†] 김 한 기,[1] 김 종 성[2‡]
[1,2]국민대학교 (대학원생, 교수)

Development and Security Analysis of GIFT-64-Variant That Can Be Efficiently Implemented by Bit-Slice Technique*

Seungjun Baek,[1†] Hangi Kim,[1] Jongsung Kim[2‡]
[1,2]Kookmin University (Graduate student, Professor)

요 약

GIFT는 CHES 2017에서 제안된 PRESENT-like 암호 알고리즘이며, 비트 슬라이스로 구현 가능한 S-box를 사용했다[1]. 선형연산으로는 Bit-permutation을 사용했기 때문에 하드웨어에서 효율적으로 구현할 수 있지만, 소프트웨어상의 비트 슬라이스 구현을 위해서는 특정 변환 과정을 거쳐야 하므로 큰 비용이 소요된다. 본 논문에서는 효율적인 비트 슬라이스 구현이 가능한 Bit-permutation과 그를 적용한 GIFT-64-variant를 제안한다. GIFT-64-variant는 차분, 선형 분석 관점에서 기존 GIFT보다 안전성이 향상되었다.

ABSTRACT

GIFT is a PRESENT-like cryptographic algorithm proposed in CHES 2017 and used S-box that can be implemented through a bit-slice technique[1]. Since bit-permutation is used as a linear layer, it can be efficiently implemented in hardware, but bit-slice implementation in software requires a specific conversion process, which is costly. In this paper, we propose a new bit-permutation that enables efficient bit-slice implementation and GIFT-64-variant using it. GIFT-64-variant has better safety than the existing GIFT in terms of differential and linear cryptanalysis.

Keywords: GIFT, Block cipher, Bit-slice, Bit-permutation, MILP, Active S-box

I. 서 론

비트 슬라이스(bit-slice) 기법은 1997년 Biham이 DES에 처음 적용하며 제안했다[2]. 이 기법을 소프트웨어상의 구현에 적용하면 n-비트 레지스터 환경에서 한 번의 논리 명령을 통해 n번의 논리 명령을 동시에 실행할 수 있다. 따라서 소프트웨어 구현 측면에서 병렬 처리가 가능하게 되므로 높은 효율성을 갖게 되며, 이는 처리 데이터양이 적어 한 블록 암호화 효율성이 중요한 경량 환경에서 큰 장점이 된다. 또한 타이밍 공격과 같은 부채널 공격에도 더 높은 안전성을 갖게 된다[3]. Fantomas, Robin[4], Rectangle[5], PRIDE[6], RoadRunneR[7] 등의 암호 알고리즘에 비트 슬라이스 기법이 적용되었다.

일반적으로 블록암호의 선형연산으로는 Binary matrix, MDS 코드, Bit-permutation 방법이 사용된다. 이 중 Binary matrix와 MDS 코드는 행렬 곱을 사용하며 특히 후자는 유한체 연산을 바탕

Received(03. 30. 2020), Accepted(05. 20. 2020)
* 본 연구는 고려대 암호기술 특화연구센터(UD170109ED)를 통한 방위사업청과 국방과학연구소의 연구비 지원으로 수행되었습니다.
† 주저자, hellosj3@kookmin.ac.kr
‡ 교신저자, jskim@kookmin.ac.kr(Corresponding author)

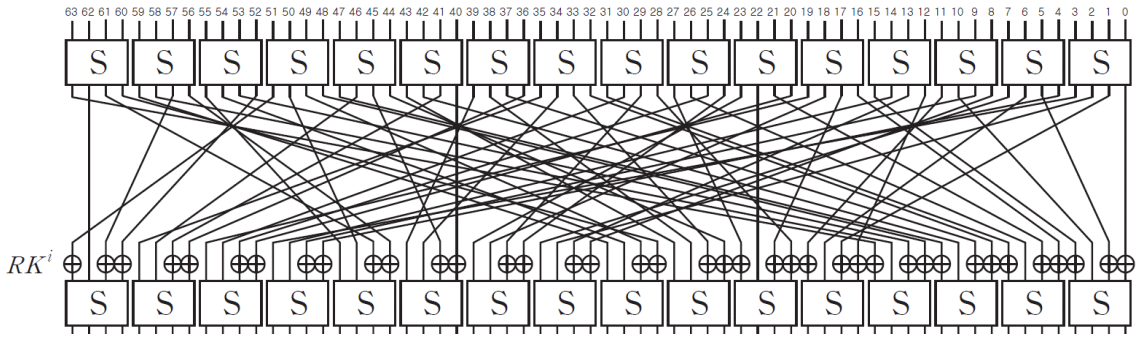

Fig. 1. Round function of GIFT-64

으로 최대의 확산 효과를 제공한다. 또한 Bit-permutation은 하드웨어 구현에서 매우 효율적이다. 하지만 비트 단위의 위치변환 연산이기 때문에 레지스터 단위의 연산으로 이루어지는 소프트웨어 구현에서는 비효율적이다.

GIFT의 제안 논문에서는 S-box에 대한 비트 슬라이스 기법을 제시했다[1]. Bit-permutation에 대해서도 레지스터 단위의 연산을 적용할 수 있는 방법을 제시했으나 많은 연산이 필요한 SWAPMOVE 기술을 적용해야 가능하다. 따라서 Bit-permutation에 대해서도 S-box에서와같이 효율적으로 소프트웨어 구현이 가능하도록 하는 연구가 필요하다.

본 논문의 구성은 다음과 같다. 2장에서는 GIFT Specification과 BOGI 논리를 설명한다. 3장에서는 효율적인 소프트웨어 구현이 가능한 새로운 Bit-permutaion을 소개하고 구성 방법, 안전성 및 효율성 분석 등을 제시한다. 마지막으로 4장에서는 본 논문에 대한 결론을 맺는다.

II. GIFT

GIFT는 Banik 등에 의해 2017년 제안되었고 2007년 제안된 PRESENT와 유사하게 SPN (Substitution Permutation Network) 구조를 지닌다. 64-비트 평문일 때 GIFT-64, 128-비트 평문일 때 GIFT-128이라고 지칭하며 모두 128-비트 키를 사용한다. 본 논문에서는 GIFT-64에 초점을 맞춘다.

2.1 GIFT Specification

GIFT의 라운드 함수는 SubCells, PermBits, AddRoundKey 세 가지의 과정으로 이루어져 있으며 총 28-라운드를 진행한다(Fig. 1). SubCells는 각 니블 단위로 Fig. 2의 S-box를 16번 적용해 혼돈을 주는 과정이다. PermBits는 Fig. 3의 Bit-permutation을 통해 과정 전체에 확산을 주는 과정이다. AddRoundKey는 PermBits 과정 후 일부 비트에 서브키를 XOR 하는 과정이다.

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

Fig. 2. Specifications of GIFT S-box

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

Fig. 3. Specifications of GIFT-64 bit-permutation

2.2 BOGI 논리

BOGI (Bad Output must go to Good Input)논리는 S-box의 출력 비트가 차분 혹은 선형 분석 관점에서 1개만 활성화된 경우 다음 S-box에서는 반드시 2개 이상의 활성화된 출력 비트를 갖도록 입력 비트를 구성하는 논리이다. 즉, BOGI 논리를 따르는 암호에서는 한 라운드의 특정 S-box의

Table 1. 1-1 bit DDT(left), LAT(right) of GIFT

ΔI / ΔO	1	2	4	8
1	0	0	0	2
2	0	0	0	0
4	0	0	0	0
8	0	0	0	0

ΔI / ΔO	1	2	4	8
1	0	0	2	4
2	0	0	0	2
4	0	0	0	0
8	0	0	0	0

출력이 나쁜 출력인 경우 해당 출력 비트는 반드시 다음 라운드의 특정 S-box의 좋은 입력이 된다.

PRESENT의 S-box와 비교했을 때 GIFT의 S-box는 더 적은 비용으로 구현할 수 있는 장점이 있지만, 차분, 선형 분석에 대한 암호학적 안전성은 더 낮다. 예를 들어, PRESENT S-box의 DBN (Differential Branch Number)은 3, LBN (Linear Branch Number)은 2인 반면 GIFT S-box의 DBN, LBN은 모두 2이다. GIFT의 설계자들은 S-box의 암호학적 안전성을 보완할 방법으로서 S-box와 Bit-permutation의 조합을 통한 BOGI 논리를 제안했다.

PermBits는 S-box의 낮은 안전성을 보완하기 위해 설계되었으며, 이 과정을 통해 각 Sbox의 입력, 출력 비트에 BOGI 논리가 적용되는 BOGI permutation을 구성할 수 있다. BOGI permutation을 적용하면 차분, 선형 분석 관점에서 라운드별 S-box가 1개씩만 활성화된 경로를 구성할 수 없으므로 암호학적으로 더 높은 안전성을 가진다.

III. GIFT-64-variant의 Bit-permutation

본 장에서는 기존 GIFT의 구현상의 한계점을 설명하고 이를 극복하기 위한 레지스터 단위의 연산을 적용할 수 있는 새로운 Bit-permutation을 제안한다. 또한 MILP 프로그램을 통해 새로운 Bit-permutation이 차분, 선형 분석 관점에서 기존 GIFT의 BOGI permutation보다 높은 안전성을 가진다는 것을 보이고, 소프트웨어 구현 수치를 제시한다.

3.1 GIFT의 소프트웨어 구현의 한계점

기존 GIFT 알고리즘 설계자들은 제안 논문에서 S-box에 대한 비트 슬라이스 기법을 제시한다. Bit-permutation에 대해서는 SWAPMOVE를 통한 소프트웨어 구현 방법을 제시하지만, Packing을 통해 데이터를 비트 슬라이스 모드로 변환하는 과정, 마지막 라운드 후 Unpacking을 통해 재변환하는 과정이 필요하다. 이에 3.2절에서는 Bit-permutation 부분에 추가적인 연산 없이 효율적인 소프트웨어 구현이 가능하도록 하는 새로운 Bit-permutation을 제시한다.

3.2 새로운 Bit-permutation 구성 단계

새로운 Bit-permutation을 찾기 위해 소프트웨어 구현에서 효율적으로 구현할 수 있는 비트 회전과 행 변환 과정을 적용했다. 새로운 Bit-permutation은 추가적인 연산 없이 알고리즘 전반의 비트 슬라이스 구현이 가능한 것을 찾는 데 목적을 뒀다.

3.2.1 레지스터 배치

먼저 최하위 비트가 있는 S-box부터 최상위 비트

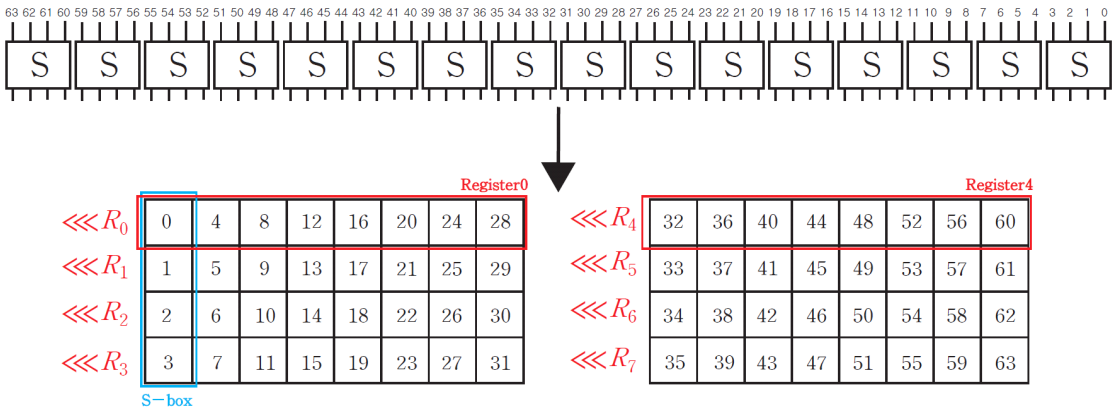

Fig. 4. The process of distributing input bits to eight registers

가 있는 S-box까지 Fig. 4와 같이 순서대로 레지스터에 배치한다.

3.2.2 비트 회전 과정

GIFT-64-variant의 Bit-permutation은 왼쪽 비트 회전, 행 변환 과정으로 이루어진다. 먼저 왼쪽 비트 회전은 레지스터 내에서 이루어지며 Register0을 고정하고($R_0 = 0$) 나머지 7개의 레지스터에 적용한다. 같은 횟수의 비트 회전이 적용되는 레지스터가 있으면 차분이 다음 라운드의 같은 S-box로 들어가는 상황이 발생한다. 따라서 넓은 확산을 위해 Register 1-7은 모두 다른 회전 횟수를 갖도록 조정했고, 각각 최소 1번에서 7번까지 왼쪽 비트 회전할 수 있다.

3.2.3 행 변환 과정

행 변환 과정은 레지스터 간에 위치를 바꾸는 과정이다. 8-비트 레지스터를 통해 Bit-permutation을 구성할 때 왼쪽 비트 회전만 적용하면 Fig. 4의 우측 8개 S-box, 좌측 8개 S-box의 차분이 각각 독립적으로 확산된다. 이를 피하고 더 폭넓은 확산을 주기 위해 레지스터 간의 행 변환은 반드시 필요하다. 하지만 행 변환이 서로 불가능한 레지스터들도 존재한다. 예를 들어 Register0과 Register3을 교환하거나 Register0과 Register7를 교환하면 Table 1의 1-1 DDT, LAT Table에 의해 BOGI 논리를 따르지 않게 된다. 이때 차분, 선형 분석 관점에서 활성 S-box 개수의 하한은 라운드 수와 동일해지므로 안전성은 낮아질 수밖에 없다. 따라서 행 변환을 하되 BOGI 논리를 따르는 범위 안에서 해야 한다. 또한 행 변환 과정은 비트 회전 과정의 추가 과정이라 볼 수 있기 때문에 최소한의 횟수만 행 변환 과정을 거치도록 해야 한다.

3.3 MILP를 이용한 활성 S-box 개수의 하한 탐색

MILP (Mixed Integer Linear Programming) Solver를 통해 활성 S-box 개수의 하한을 찾는 방법은 Mouha 등에 의해 최초로 제안됐다[8]. 이후 Sun 등이 제안한 방법을 통해 비트 단위로 차분 경로를 추정하여 더욱 정확한 차분의 확산 양상을 확인할 수 있게 됐다[9]. MILP는 선형 부등식 형태의 제약식과 최댓값 혹은 최솟값을 찾아주는 목적함수로 구성되며 MILP solver를 통해 최적해를 찾을 수 있다. 본 논문에서는 MILP solver로 CPLEX를 사용하였고[10], [9]에서 제안한 방법을 통해 새로운 Bit-permutation을 적용한 GIFT-64-variant의 안전성을 분석했다.

GIFT S-box의 제약식을 구하기 위해 먼저 SageMath를 이용하여 DDT, LAT 각각에 대한 S-box의 컨벡스 헐(convex hull)을 찾는 과정이 필요하다[11]. 이후 그리디 알고리즘(greedy algorithm)을 사용하여 각각 237, 211개 선형 부등식으로 이루어져 있던 DDT, LAT 컨벡스 헐 중 24, 19개 부등식을 선택할 수 있다. 이를 통해 Bit-permutation을 고려한 제약식들과 목적함수를 CPLEX 프로그램에 입력하면 차분, 선형 분석 관점에서 활성 S-box 개수의 하한을 탐색할 수 있다.

3.4 제안 방법을 통해 구성한 Bit-permutation과 GIFT-64-variant

기존 GIFT는 3라운드 Full-diffusion을 만족하지만, Bit-permutation을 왼쪽 비트 회전과 행 변환을 통해 구성할 때 3라운드 Full-diffusion을 달성하기는 불가능했다. 4라운드 Full-diffusion을 만족하더라도 차분, 선형 관점에서 충분한 안전성을 가질 수 있기 때문에 모든 조합 중 4라운드 Full-diffusion을 만족하는 경우의 수를 찾았다.

3.2.2의 방법에 따라 레지스터1은 고정하고 나머지 7개 레지스터에서 왼쪽 비트 회전 연산이 가능한 경우의 수는 5,040(7!)가지이다. 또한 3.2.3의 방법에 따라 BOGI 논리를 만족하지 않거나 Register0-3, Register4-7 내에서만 교환하는 상황을 배제한다. 이때 행 변환이 가능한 가짓수는 10가지이다. 따라서 전체 조합의 경우의 수는 총 50,400가지이며, 이 중 27,360가지 경우가 4라운드 Full-diffusion을 만족한다. CPLEX 프로그램을 통해 모든 경우의 수를 전수조사하여 가장 높은 안전성을 갖는 조합을 찾을 수 있었다.

Fig. 5는 왼쪽 비트 회전과 행 변환 과정을 통해 본 논문에서 제시하는 새로운 Bit-permutation을 구성하는 과정이다. 왼쪽 비트 회전은 $(R_0, R_1, R_2, R_3, R_4, R_5, R_6, R_7) = (0, 2, 7, 3, 1, 5, 6, 4)$, 행 변환은 Register3, Register7 간 이루어진다. 비트 슬라이스 기법은 Fig. 6과 같이 구현할 수 있으며, 최종 Bit-permutation은 Fig. 7과 같다. S-box는 GIFT 제안 논문[1]에서 제시된 Fig. 8과 같이 비트

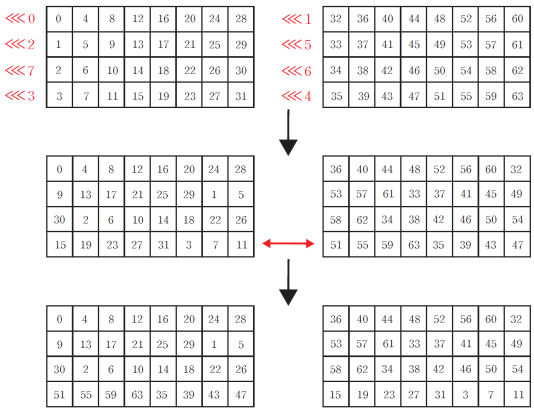

Fig. 5. Bit-permutation configuration process of GIFT-64-variant

```
/*Input: X[0]-reg0, X[1]-reg1, X[2]-reg2, X[3]-reg3
         Y[0]-reg4, Y[1]-reg5, Y[2]-reg6, Y[3]-reg7 */
X[1] = (X[1] << 2) | (X[1] >> 6);
X[2] = (X[2] << 7) | (X[2] >> 1);
Y[0] = (Y[0] << 1) | (Y[0] >> 7);
Y[1] = (Y[1] << 5) | (Y[1] >> 3);
Y[2] = (Y[2] << 6) | (Y[2] >> 2);
T = (X[3] << 3) | (X[3] >> 5);
X[3] = (Y[3] << 4) | (Y[3] >> 4);
Y[3] = T;
/*Output: X[0]-reg0, X[1]-reg1, X[2]-reg2, X[3]-reg3
         Y[0]-reg4, Y[1]-reg5, Y[2]-reg6, Y[3]-reg7 */
```

Fig. 6. Bit-slice implementation of GIFT-64-variant bit-permutation

슬라이스 구현하며, 이를 종합적으로 GIFT에 적용한 GIFT-64-variant의 라운드 함수는 Fig. 9과 같다.

3.5 GIFT-64-variant 안전성 및 효율성 분석

GIFT-64-variant의 활성 S-box 개수의 하한은 기존 GIFT보다 차분 관점에서는 4, 7라운드에서 1개씩 증가했고, 선형 관점에서는 6, 7, 8라운드에서 1개씩 증가했다(Table 2). 또한 분석 결과 9라운드 기준

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

Fig. 7. Specifications of GIFT-64-variant bit-permutation

```
/* Input: (MSB) X[3], X[2], X[1], X[0] (LSB) */
X[1] = X[1] ^ (X[0] & X[2]);
T = X[0] ^ (X[1] & X[3]);
X[2] = X[2] ^ (T | X[1]);
X[0] = X[3] ^ X[2];
X[1] = X[1] ^ X[0];
X[0] = ~X[0];
X[2] = X[2] ^ (T & X[1]);
X[3] = T;
/* Output: (MSB) X[3], X[2], X[1], X[0] (LSB) */
```

Fig. 8. Bit-slice implementation of GIFT S-box

최상의 차분 경로의 확률은 $2^{-35.4150}$, 최상의 선형 경로의 전체 correlation은 2^{-18}임을 알 수 있었다.

소프트웨어상의 수치에서도 유의미한 차이가 있었고 결과는 Table 3과 같다. 일반적인 범용 PC에서 실험하였고, 세부 사양으로 프로세서는 AMD Ryzen 5 3600(3.6GHz), 메모리는 32GB, 운영체제는 Windows 10 64bit이다. 컴파일러는 Visual Studio 2019 C/C++(ver 16.4.5)를 사용했다. GIFT-64는 github에 올라온 israelqwe의 코드에 비트 슬라이스 기법이 적용되도록 수정하여 사용했고[12], GIFT-64-variant는 자체 구현하여 사용했다.

Table 2. Comparison of the lower bounds of the number of active S-boxes

Cipher	DC/LC	Round								
		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
GIFT-64 -variant	DC	1	2	3	6	7	10	14	16	18
	LC	1	2	3	5	7	10	13	16	18

Table 3. Comparison of the speed of implementations for single block encryption

Cipher	Cycle per byte
GIFT-64	110.25 c/b
GIFT-64-variant	51.75 c/b

3.6 기타

지금까지 8-비트 레지스터를 이용하여 새로운 Bit-permutation을 제시했다. 이와 별개로 16-비

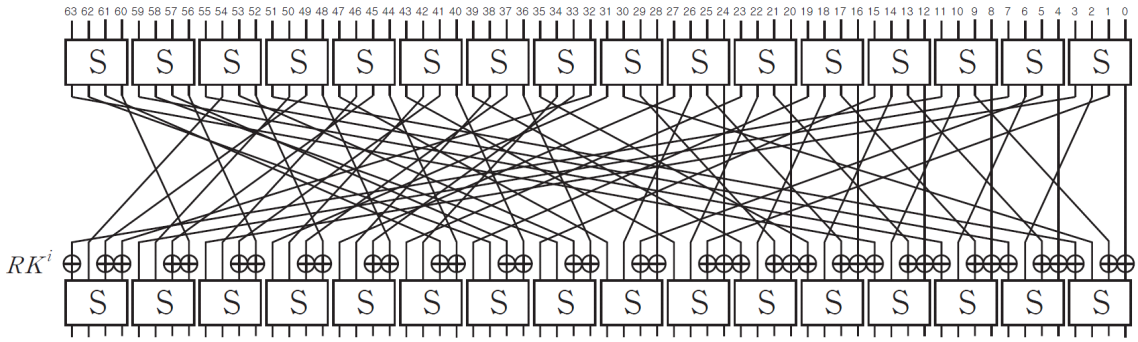

Fig. 9. Round function of GIFT-64-variant

트 레지스터를 통해서도 효율적인 소프트웨어 구현이 가능한 Bit-permutation을 구성할 수 있는지 조사했다. 3.2절의 방법과 같으며 8-비트 버전과 달라진 점은 8개를 사용했던 레지스터를 4개만 사용한다는 점이다. 16-비트 레지스터를 사용할 때는 8-비트 레지스터 때와 같이 Register0을 고정하고 나머지 3개 레지스터가 왼쪽 비트 회전 연산이 가능한 경우를 고려한다. 이때는 행 변환 과정 없이도 4라운드 Full-diffusion을 만족하는 경우의 수를 찾을 수 있으며, 총 2730가지 경우의 수 중 96가지가 이에 해당했다. 하지만 전수조사 결과 차분, 선형 분석 관점에서 기존 GIFT의 안전성에 미치지 못한다는 사실을 알 수 있었고, 행 변환을 추가로 여러 번 적용하더라도 유의미한 차이는 관찰할 수 없었다.

IV. 결 론

본 논문에서는 8-비트 레지스터 단위의 연산이 가능하고, 경량 환경인 8-비트 microprocessor 구현에 최적화된 GIFT-64의 새로운 Bit-permutation을 제안했다. 이를 통해 기존의 Bit-permutation을 사용했을 때보다 차분, 선형 관점에서 더 높은 활성 S-box 개수의 하한을 가지게 만들 수 있었다. 효율성 관점에서도 약 2.13배의 속도 향상 결과를 얻을 수 있었다.

최근 사물인터넷 환경이 발달하면서 기기는 점점 경량화되어가고 있으며, 이에 따라 내부의 암호 알고리즘을 낮은 비용으로 구현할 수 있어야 한다. 안전성과 효율성을 일정 기준 이상 만족한다면 소프트웨어상에서는 경량 기기 암호 알고리즘을 비트 슬라이스를 통해 구현하는 것이 가장 좋다. 향후 SPN 구조를 갖는 PRESENT-like 암호 알고리즘이나 다른 구조를 갖는 암호 알고리즘들까지도 본 논문에서 제시하는 방식과 유사하게 Bit-permutation을 변환하면 더 안전하고 효율적인 암호 알고리즘을 만들 수 있을 것으로 예상한다.

References

[1] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim and Yosuke Todo, "GIFT: A Small Present Towards Reaching the Limit of Lightweight Encryption," CHES'17, LNCS 10529, pp. 321-345, Sep. 2017.

[2] Eli Biham, "A Fast New DES Implementation in Software," FSE'97, LNCS 1267, pp. 260-272, Jan. 1997.

[3] Mitsuru Matsui and Junko Nakajima, "On the Power of Bitslice Implementation on intel Core2 Processor," CHES'07, LNCS 4727, pp. 121-134, Sep. 2007.

[4] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varici, "LS-Designs: Bitslice Encryption for Efficient Masked Software Implementations," FSE'14, LNCS 8540, pp. 18-37, Mar. 2014.

[5] WenTao Zhang, ZhenZhen Bao, DongDai Lin, Vincent Rijmen, Bohan Yang and Ingrid Verbauwhede, "RECTANGLE: A Bit-lice Lightweight

Block Cipher Suitable for Multiple Platforms," Science China Information Sciences 58(12), pp. 1-15, Nov. 2015.

[6] Martin R. Albrecht, Benedikt Driessen, Elif Bilge Kavun, Gregor Leander, Christof Paar and Tolga Yalçın, "Block Ciphers - Focus On The Linear Layer(fear. PRIDE)," CRYPTO'14, LNCS 8616, pp. 57-76, Aug. 2014.

[7] Adnan Baysal and Sühap Şahin, "RoadRunnerL: A Small And Fast Bitslice Block Cipher For Low Cost 8-bit Processors," LightSec'15, LNCS 9542, pp. 58-76, Sep. 2015.

[8] Nicky Mouha, Qingju Wang, Dawu Gu and Bart Preneel, "Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming," Inscrypt'11, LNCS 7537, pp. 57-76, Dec. 2011.

[9] Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, Ling Song, "Automatic Security Evaluation and (Related-key) Differential Characteristic Search: Application to SIMON, PRESENT, LBlock, DES(L) and Other Bit-Oriented Block Ciphers," ASIACRYPT'14, LNCS 8873, pp. 158-178, Dec. 2014.

[10] CPLEX, IMB, https://www.ibm.com/kr-ko/analytics/cplex-optimizer, Visited on Feb 3. 2020.

[11] System for Algebra and Geometry Experimentation, Stein A. William, https://www.sagemath.org/, Visited on Feb 4. 2020.

[12] gift-cipher, israelqwe, https://github.com/israelqwe/gift-cipher, Visited on March 8. 2020.

〈저자소개〉

백 승 준(Seungjun Baek) 학생회원
2019년 2월: 국민대학교 수학과 졸업
2020년 3월~현재: 국민대학교 금융정보보안학과 석사과정
〈관심분야〉 정보보호, 암호 알고리즘

김 한 기(Hangi Kim) 학생회원
2016년 2월: 국민대학교 수학과 졸업
2018년 2월: 국민대학교 금융정보보안학과 석사
2018년 3월~현재: 국민대학교 금융정보보안학과 박사과정
〈관심분야〉 정보보호, 암호 알고리즘

김 종 성(Jongsung Kim) 종신회원
2000년 8월/2002년 8월: 고려대학교 수학 전공 학사/이학석사
2006년 11월: K.U.Leuven. ESAT/SCD-COSIC 정보보호 전공 공학박사
2007년 2월: 고려대학교 정보보호대학원 공학박사
2007년 3월~2009년 8월: 고려대학교 정보보호기술연구센터 연구교수
2009년 9월~2013년 2월: 경남대학교 e-비즈니스학과 조교수
2013년 3월~2017년 2월: 국민대학교 수학과 부교수
2014년 3월~현재: 국민대학교 일반대학원 금융정보보안학과 부교수
2017년 3월~현재: 국민대학교 정보보안암호수학과 부교수
〈관심분야〉 정보보호, 암호 알고리즘, 디지털 포렌식