Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.26, NO.6, Dec. 2016 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2016.26.6.1421

ARX 구조를 가지는 블록 암호에 대한 효율적인 차분 경로 자동 탐색 알고리즘*

김 서 진,[1†] 강 형 철,[1] 홍 득 조,[2‡] 성 재 철,[3] 홍 석 희[1]
[1]고려대학교 정보보호대학원, [2]전북대학교 IT 정보공학과, [3]서울시립대학교 수학과

Efficient Differential Trail Searching Algorithm for ARX Block Ciphers*

Seojin Kim,[1†] HyungChul Kang,[1] Deukjo Hong,[2‡] Jaechul Sung,[3] Seokhie Hong[1]
[1]Graduate School of Information Security, Korea University
[2]Department of Information Technology, Chonbuk National University
[3]Department of Mathematics, University of Seoul

요 약

본 논문에서 우리는 ARX 구조를 가지는 블록 암호에 대한 차분 경로 탐색을 효율적으로 수행하는 방법에 대해 제안한다. 우리는 두 가지 기법을 이용하여 A. Biryukov 등이 제안한 차분 경로 자동 탐색하는 알고리즘을 최적화하였고, 이를 블록 암호 SPECK에 적용하여 Birykov의 결과보다 2~3배 향상된 결과를 얻었다. 이는 ARX 구조를 가지는 블록 암호에 대한 기제안된 결과보다 더 좋은 차분 경로를 찾는데 도움을 줄 수 있다.

ABSTRACT

In this paper, we suggest an advanced method searching for differential trails of block cipher with ARX structure. we use two techniques to optimize the automatic search algorithm of differential trails suggested by A. Biryukov et al, and obtain 2~3 times faster results than Biryukov's when implemented in block cipher SPECK. This results contribute to find better differential trails than previous results.

Keywords: ARX structure, Differential trails, Automatic search algorithm, SPECK

I. 서 론

1990년, E. Biham과 A. Shamir가 DES에 대한 차분 공격(Differential Analysis)[1]을 제안한 이후로, 차분 공격은 블록 암호에 대한 공격 중 가장 중요한 공격으로 다뤄지고 있다. 차분 공격은 비선형 구조인 S-box에 대한 입력 차분과 이에 대응되는 출력 차분을 이용하여 수행되며, 이를 위하여 모든 입·출력 차분에 대해서 차분분포표(Difference Distribution Table)를 생성한다.

ARX(Addition, Rotation and eXclusive-or) 구조는 법 덧셈, 순환이동, XOR로 이루어져 있다. ARX 구조는 단순하기 때문에 다양한 경량 블록 암호에 적용되고 있다[6, 7].

이 구조에서는 법 덧셈(modular addition)이 S-box와 같은 비선형 구조가 되며, 이 구조에 대한 차분 분석을 위해서는 법 덧셈의 입·출력 차분 확률을 계산해야 한다. 이때, 하나의 입력 차분과 하나의 출력 차분을 고려해야하는 기존의 S-box와는 달리, 법 덧셈은 두 개의 입력 차분과 하나의 출력 차분을 고려해야 한다. 또한, 입·출력 크기도 일반 S-box보

Received(09. 23. 2016), Modified(10. 26. 2016), Accepted(10. 26. 2016)
* 본 논문은 2016년도 하계학술대회에 발표한 우수논문을 개선 및 확장한 것임
† 주저자, canonicus@naver.com
‡ 교신저자, deukjo.hong@jbnu.ac.kr(Corresponding author)

다 크기 때문에 더 많은 연산을 필요로 한다. 따라서 이러한 ARX 구조에 대한 차분 공격은 다른 방법으로 진행되어야 하며, Biryukov 등이 이에 대해 새로운 방법을 제안했다[2]. [2]에서 제안된 방법도 효율적인 방법이나 분석 라운드 수가 증가할수록 분석 시간이 기하급수적으로 증가하는 단점이 있다.

본 논문에서는 Biryukov 등이 제안한 방법을 실행 시간 측면에서 개선시키는 방법을 제안한다. 실행 시간을 단축시키기 위해 새롭게 제안하는 방법을 이용하였다.

본 논문은 다음과 같이 구성된다. 2장에서 법 덧셈 차분의 주요 특징과, 공격을 수행하는 블록 암호 SPECK에 대해 설명한다. 3장에서는 Biryukov 등이 제안한 방법을 설명하고, 4장에서 이 방법을 실행 시간 측면에서 개선시키는 방법을 제안한다. 마지막으로 5장에서 결론을 맺는다.

II. 관련 연구

2.1 표기법

본 논문에 사용된 표기법은 다음과 같다.

- o $\oplus$: XOR(eXclusive-Or) 연산
- o $\wedge$: And 연산
- o $\neg$: 보수
- o $x|y$: 두 비트열 x와 y의 연접
- o $x \gg i$ ($x \ll i$): i만큼 오른쪽으로 (왼쪽으로) 비트 이동 연산
- o $x \ggg i (x \lll i)$: i만큼 오른쪽으로 (왼쪽으로) 비트 순환이동 연산
- o w: 워드 크기
- o a_i: a의 오른쪽에서 i번째 비트
- o $x[i:1]$: 비트열 x의 하위 i개 비트

2.2 블록 암호 SPECK

SPECK은 2013년, NSA가 제안한 ARX 구조를 가지는 경량 블록 암호로서, 블록 크기에 따라 총 5가지 종류로 나뉘며, 각각 다양한 키 크기를 지원한다[3].

SPECK의 한 라운드 구조는 Fig. 1과 같다. 본 논문에서는 블록 크기 32-비트와 48-비트를 가지는 SPECK32과 SPECK48에 대해서 분석한다.

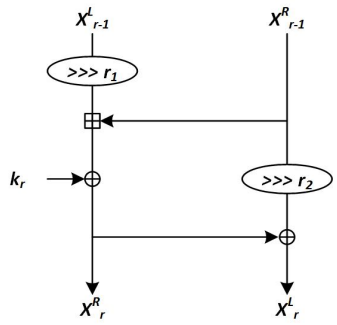


Fig. 1. Round Function of SPECK

Fig. 1에서 X_L과 X_R은 워드를 의미하며, SPECK32에서는 각각 16-비트, SPECK48에서는 각각 24-비트이다. 그리고 k_r은 r 라운드 키이다.

r 라운드를 수식으로 표현하면 다음과 같다.

$$X_L^r = ((X_L^{r-1} \ggg r_1) \boxplus X_R^{r-1}) \oplus k_r$$
$$X_R^r = (X_R^{r-1} \lll r_2) \oplus X_L^r$$

순환이동 상수인 r_1, r_2는 SPECK32에서는 $r_1 = 7$, $r_2 = 2$이고 나머지 SPECK에서는 $r_1 = 8$, $r_2 = 3$이다.

SPECK의 키스케쥴은 차분 경로를 찾는 과정과는 관련이 없으므로, 자세한 설명은 생략한다. 더 자세한 사항은 [3]에서 확인할 수 있다.

2.3 법 덧셈 차분 특징

법 덧셈에 대한 XOR 차분은 S-box와 달리 두 개의 입력 차분과 하나의 출력 차분으로 이루어져 있다. S-box에서는 하나의 입력 차분과 이에 대응되는 출력 차분이 나올 확률은 전수조사를 통해 구할 수 있으며, 이때, 입력 비트가 n-비트인 경우 2^{n+1}번의 계산이 필요하다. 법 덧셈의 경우, 입력 차분이 두 개이기 때문에 전수조사를 통해 확률을 계산한다

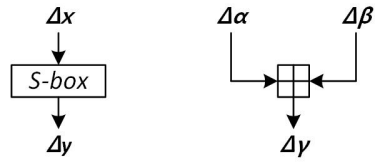


Fig. 2. Input/Output difference on S-box and modular addition

면 2^{2n+2}번의 계산이 필요하다. 여기서 n의 값이 커질수록 더 많은 계산량이 필요하게 되는데, 이 계산 복잡도를 n으로 낮춰주는 방법을 H. Lipmaa가 제안했다[4].

먼저 입력 차분이 α, β이고, 출력 차분이 γ일 때, 법 덧셈에 대한 XOR 차분 확률(eXclusive-or Differential Probability; xdp)을 다음과 같이 정의한다.

정의 1. [4] 입력 차분이 α, β이고, 출력 차분이 γ일 때, 법 덧셈에 대한 XOR 차분 확률은 다음과 같다:

$$xdp(\alpha,\beta\rightarrow\gamma)=\Pr[((x\oplus\alpha)+(y\oplus\beta))\oplus(x+y)=\gamma]$$

xdp는 법 덧셈에서 발생하는 자리올림과 자리올림의 차분을 살펴봄으로써 간단하게 구할 수 있다.

정리 1. [4] 입력 차분이 α, β이고, 출력 차분이 γ일 때, 만약 $x,y\in\{0,1\}^w$에 대해서 $carry(x,y):=c\in\{0,1\}^w$이면, $xdp(\alpha,\beta\rightarrow\gamma)=\Pr[carry(x,y)\oplus carry(x\oplus\alpha,y\oplus\beta)=(\alpha\oplus\beta\oplus\gamma)]$. 이다. 여기서, $c_0=0$ 이고 $c_{i+1}:=(x_i\wedge y_i)\oplus(x_i\wedge c_i)\oplus(y_i\wedge c_i)$ 이다.

즉, xdp는 각 자리를 계산할 때 $carry$, 즉 한 비트를 살펴보기 때문에 n-비트에 대해서 n의 계산복잡도가 필요하다.

정리 2. [4] 입력 차분이 α, β이고, 출력 차분이 γ일 때, 가능한 모든 차분(확률이 0이 아닌 모든 차분)은 다음과 그 역이 성립한다.

$eq(\alpha\ll 1,\beta\ll 1,\gamma\ll 1)\wedge((\alpha\oplus\beta\oplus\gamma)\oplus(\alpha\ll 1))=0$

여기서, $eq(x,y,z):=(\neg x\oplus y)\wedge(\neg x\oplus z)$이다.

(즉, $eq(x,y,z)_i=1\Leftrightarrow x_i=y_i=z_i$)

즉, 어떤 입력 차분 α,β에 대하여 γ의 출력 차분이 발생할 확률이 0이 아니면 $i\in\{0,1,...,w-1\}$에 대하여 $\alpha_{i-1}=\beta_{i-1}=\gamma_{i-1}=\alpha_i\oplus\beta_i\oplus\gamma_i$라는 것과 동치이다.

두 보조정리의 증명은 [4]를 따르고, 이 두 보조정리를 이용해 [4]에서는 xdp를 구하는 알고리즘을 제안한다.

Algorithm 1. [4] Log-time Algorithm for xdp

Input: $\delta=(\alpha,\beta\rightarrow\gamma)$

Output: $xdp(\delta)$

1. If $eq(\alpha\ll 1,\beta\ll 1,\gamma\ll 1)\wedge((\alpha\oplus\beta\oplus\gamma)\oplus(a\ll 1))\neq 0$, then return 0;
2. Return $2^{-W_h(\neg eq(\alpha,\beta,\gamma)\wedge(2^w-1))}$

$W_h(x)$: x의 해밍웨이트

III. Biryukov 등이 제안한 알고리즘[2]

차분 공격 과정은 비선형 연산에 대한 차분분포표를 만들면서 시작한다. S-box의 경우, 입력 차분이 n-비트이고, 출력 차분이 m-비트 인 경우, 차분분포표의 원소 개수는 2^{n+m}개가 된다. 법 덧셈의 경우 입력 차분이 n-비트 인 경우, 출력 차분도 n-비트가 되고, 차분분포표의 원소 개수는 2^{3n}개가 된다. 입·출력 차분이 각각 16-비트 이상이면 차분분포표가 2^{48}개 이상의 원소를 가지므로, 32TB 이상의 메모리가 필요하다. 따라서, Biryukov 등은 차분분포표를 생성하지 않고 경로를 탐색하는 방법을 제안했다[2].

정리 3. [2] XOR 차분 확률은 입·출력 차분의 크기가 커짐에 따라 줄어든다. 따라서 α, β, γ의 입출력 차분이 있을 때 다음이 성립한다.

$p_w\leqq p_{w-1}\leqq p_{w-2}\cdots\leqq p_1$,

where $p_i=xdp(\alpha[i:1],\beta[i:1]\rightarrow\gamma[i:1])$

Biryukov 등은 명제 1.을 이용하여 한 라운드에 한 비트 씩 추가하며 재귀적으로 구하는 방법을 사용한다. r 라운드의 정해진 입력 차분에 대한 출력 차분을 계산할 때, 출력 차분에 한 비트 씩 붙여가며 xdp를 계산한다. 계산한 xdp가 기준 확률(bound)보다 높을 경우에는 한 비트를 추가로 붙이거나, 모든 비트에 대해서 계산이 끝났을 때에는 $r+1$ 라운드를 계산한다. 1 라운드의 경우, 입력 차분도 추가로 계산하여야 한다.

여기서 기준 확률은 이미 구해진 r 라운드까지의 최대 확률 B_r을 이용한다. 첫 번째 라운드에서는 입·출력 차분에 대해 차분 확률 xdp_1이 $xdp_1\times B_{r-1}\geqq B_r$을 만족시키는 모든 차분 쌍을 고려한다. $j(2<j<r)$ 라운드부터는 $xdp_1\times xdp_2\times\cdots\times xdp_j\times B_{r-j}\geqq B_r$을 만족시키는 출력 차분을 고려한다. 그리고 마지막 라운

드에서는 $xdp_1 \times xdp_2 \times \cdots \times xdp_r \geqq B_r$을 만족시키는 출력 차분을 찾고, 해당 출력 차분이 있을 경우에는 새로운 확률과 차분 경로를 저장한다.

원하는 라운드 r에 대해 B_r을 알지 못하는 경우에는 B_{r-1}을 B_r로 대체하고, 경로를 탐색하지 못 한다면 $B_r \leftarrow B_r \times 1/2$의 연산을 수행해주면서 B_r을 구해준다. Biryukov 등이 SPECK에 적용한 알고리즘은 부록에서 찾을 수 있다(부록 A. Algorithm 2).

알고리즘의 결과는 다음 표와 같다.

Table 1. Searching time of Algorithm 2〔2〕 (Intel Core™ E5-2637 CPU 3.50GHz)

r	SPECK32		SPECK48	
	Prob.($\log_2(p)$)	time	Prob.($\log_2(p)$)	time
1	0	0sec	0	0sec
2	-1	0sec	-1	0sec
3	-3	0sec	-3	0sec
4	-5	0sec	-6	0sec
5	-9	0sec	-10	1sec
6	-13	1sec	-14	3sec
7	-18	1min	-19	1min
8	-24	34min	-	

IV. 제안하는 알고리즘

본 장에서는 〔2〕에서 제안한 알고리즘의 계산 복잡도를 줄이는 방법에 대해서 소개한다. 본 논문에서는 계산 복잡도를 줄이기 위해 두 가지 방법을 사용하였다. 하나는 경로 탐색 시 확률이 0이 되는 부분을 전혀 계산하지 않는 방법이고, 나머지 하나는 경로 탐색 시 비트 단위로 확률을 계산하는 방법이다.

4.1 차분 경로 탐색 최적화

계산 복잡도를 줄이기 위해서, 본 논문에서 제안하는 방법의 핵심 아이디어는 확률이 0이 되는 부분을 미리 파악하여 계산하지 않고 버리는 것이다.

알고리즘 2(부록 A)에서는 한 비트 값 0과 1을 순서대로 붙여가며 xdp를 계산하여 확률이 0인지 아닌지를 확인한다. 이때, 0이면 return한 뒤에 다음 차분을 고려하는 방식으로 진행된다.

하지만 알고리즘 1에서 xdp가 먼저 0인지 계산하는 부분을 살펴보면, 다음과 같은 정리 4가 나올 수 있음을 알 수 있다.

정리 4. 〔5〕 $xdp(\alpha, \beta \rightarrow \gamma) \neq 0$
iff $(\alpha[0] \oplus \beta[0] \oplus \gamma[0]) = 0$ and $\alpha[i-1] = \beta[i-1] = \gamma[i-1] = \alpha[i] \oplus \beta[i] \oplus \gamma[i]$
for $\alpha[i-1] = \beta[i-1] = \gamma[i-1]$, $i \in \{0, ..., n-1\}$

정리 4에 의하면, 한 비트를 붙이기 바로 전의 비트 값($i-1$번째 비트)을 확인하면, 다음 비트가 무엇이 될지 예상할 수 있다. 예를 들어, 입력 차분 비트 α, β와 출력 차분 비트 γ의 i 번째 비트가 $(\alpha_i, \beta_i, \gamma_i) = (0,0,0)$이라면, 입·출력 차분의 $(i+1)$번째 비트로 가능한 값은 네 가지이다 - $(0,0,0)$, $(0,1,1)$, $(1,0,1)$, $(1,1,0)$.

이 방법을 이용하면, 알고리즘 2의 11, 18, 22, 34 줄에서의 xdp 계산을 줄일 수 있어, 계산 복잡도가 약 i비트 xdp계산에서 i 만큼 줄어들게 된다.

즉 이전 비트에서 $(0,0,0)$ 또는 $(1,1,1)$이 나올 경우, 중간 라운드에서 다음에 나올 경우를 한 가지로 줄일 수 있기 때문에 전체적으로는 $1/4 \times 1/2 = 1/8$(((0,0,0)또는 $(1,1,1)$이 나올 확률) $\times$ (가능한 γ 비트를 반으로 줄임))만큼 계산복잡도를 줄일 수 있다.

4.2 비트 단위 xdp 계산

만약의 r 라운드의 경로에 한 라운드를 추가하여 탐색할 경우, 하나의 라운드에서 비트를 추가할 때마다 늘어난 차분에 대해 확률을 구해야 하므로 한 라운드마다 $\sum_{k=1}^{w} k$ 번의 계산이 필요하다. 따라서 전체 r 라운드까지의 경로를 탐색하기 위해서는 $r \times w(w+1)/2$ 번의 계산이 필요하다.

$xdp(\alpha[i:1], \beta[i:1] \rightarrow \gamma[i:1])$를 알고 있을 때, $\alpha[i:1]$, $\beta[i:1]$, $\gamma[i:1]$의 앞에 한 비트 x_α, x_β, x_γ를 4.1에서와 같이 확률이 0이 되지 않게 하며 붙이는 경우를 살펴본다. 여기서 최상위 비트인 α_i, β_i, γ_i를 확인하면, $xdp(\alpha[i+1:1], \beta[i+1:1] \rightarrow \gamma[i+1:1])$를 $xdp(\alpha[i:1], \beta[i:1] \rightarrow \gamma[i:1])$에 1, 또는 1/2을 곱해주며 계산할 수 있다. 이것은 알고리즘 1에서 확률을 구하는 $2^{-W_h(\neg eq(\alpha,\beta,\gamma) \wedge (2^w-1))}$를 계산해야 할 때, $-W_h(\neg eq(\alpha,\beta,\gamma) \wedge (2^w-1))$는 $1 \leqq j < i$인 j에 대해 $\alpha_j = \beta_j = \gamma_j$를 만족하지 않는 j의 개수가 된다. 다시 말해 $xdp(\alpha[i+1:1],$

$\beta[i+1:1] \rightarrow \gamma[i+1:1]$)를 계산할 때는 $\alpha_i = \beta_i = \gamma_i$인지 확인하고, 성립한다면 1을, 성립하지 않는다면 1/2를 곱해주며 진행할 수 있다. 따라서 w-비트까지 한 비트씩 추가한다면 확률을 $\sum_{k=1}^{w} k$ 번이 아니라 $\sum_{k=1}^{w} 1$ 번의 계산으로 구할 수 있다.

즉, w-비트 워드에 r 라운드의 경로 하나를 탐색하기 위해서는 $r \times w$의 계산이 필요하다. 전체 알고리즘에서 xdp를 계산할 때, 비트 수에 따라 복잡도가 늘어나는 것이 아니라 한 번 계산에 무조건 1의 복잡도를 가지며 전체 계산 복잡도를 낮출 수 있다.

Table 2은 $(i-1)$ 번째 입·출력 차분 비트와 가능한 i 번째 입·출력 차분 비트, 그리고 i 번째 입·출력 차분 비트가 늘어날 경우 곱해지는 확률을 나타낸 것이다.

위 표를 보면 $(i-1)$ 번째 입·출력 차분 비트가 모두 같을 경우, 확률은 유지되고 i 번째 올 수 있는 비트가 네 가지 경우로 정해진다. $(i-1)$ 번째 입·출력 차분 비트가 같지 않을 경우, i 번째 올 수 있는 입·출력 차분 비트는 모든 경우가 가능하고, 이때 확률은 1/2이 곱해지게 된다. 이런 방식을 이용하여 계산한다면 차분 확률이 0이 되는 경로는 처음부터 탐색하지 않고, 확률을 비트단위로 구해주며 효율적으로 경로를 생성할 수 있게 된다.

Table 2. i^{th} bit and corresponding $(i-1)^{th}$ bit, and probability which will be multiplied

$(i-1)^{th}$ bit	i^{th} bit	Prob.
(0, 0, 0)	(0, 0, 0)	1
	(0, 1, 1)	
	(1, 0, 1)	
	(1, 1, 0)	
(0, 0, 1)	All possible	1/2
(0, 1, 0)		
(0, 1, 1)		
(1, 0, 0)		
(1, 0, 1)		
(1, 1, 0)		
(1, 1, 1)	(0, 0, 1)	1
	(0, 1, 0)	
	(1, 0, 0)	
	(1, 1, 1)	

4.3 결과

Biryukov 등은 7 라운드 이상, 또는 SPECK48 이상에서는 HPC 클러스터를 이용해 속도를 줄였다. 하지만, 본 논문에서는 동일한 환경을 구성할 수가 없어서 SPECK32에서 8라운드 이하, SPECK48에서 7라운드 이하의 속도와 비교한다.

위 표에서 알고리즘 3 수행시간을 보면, 컴퓨터 사양이 더 좋지 않음에도 불구하고, 알고리즘 2보다 2~3 배 빠른 것을 알 수 있다. 즉, 동일한 사양의 컴퓨터나 HPC 클러스터를 사용한다면 더 좋은 결과를 보일 것이다.

Table 3. Time comparison of Algorithm 2 and 3 (time: Intel Core™ E5-2637 CPU 3.50GHz time': Intel Core™ i7-2600 CPU 3.40GHz)

r	SPECK32			SPECK48		
	Prob.	time	time'	Prob.	time	time'
1	0	0sec	0sec	0	0sec	0sec
2	-1	0sec	0sec	-1	0sec	0sec
3	-3	0sec	0sec	-3	0sec	0sec
4	-5	0sec	0sec	-6	0sec	0sec
5	-9	0sec	0sec	-10	1sec	1sec
6	-13	1sec	1sec	-14	3sec	3sec
7	-18	1min	26sec	-19	1min	33sec
8	-24	34min	11min	-		

Prob. = $\log_2(p)$

V. 결 론

본 논문에서는 Biryukov 등이 제안한 차분 경로 탐색 알고리즘의 수행시간을 단축시키는 방법에 대해 제안하였다. 하지만 Biryukov 등이 제안한 알고리즘 2는 수행시간이 오래 걸려 SPECK48 이상의 알고리즘에서는 원하는 확률까지의 경로를 생성할 수 없었다. 이에 우리는 차분 확산의 특징과, 비트 단위 연산을 통하여 탐색 시간을 2~3배 단축시킬 수 있었다(알고리즘 3). 이를 Biryukov 등이 사용했던 플랫폼에서 수행한다면 이전 결과보다 더 많은 라운드에서의 탐색이 가능함을 알 수 있다.

알고리즘 3은 SPECK 뿐만 아니라 다른 ARX 구조를 가지는 블록 암호들에도 적용하면 차분 경로를 더 빠른 시간 내에 찾을 수 있을 것으로 예상된다.

VI. 부록 A

Algorithm 2. [2] Search for the Best Differential Trail in ARX (Application to SPECK)

Input - n: number of rounds
w: word size in bits
r_1, r_2: right and left rotation constant
r: current round ($n \geqq r \geqq 1$)
i: current bit position ($w > i \geqq 0$)
$B=(B_1, B_2, ..., B_{n-1})$: probabilities of the best trails for rounds 1, 2, ..., (n-1) (global)
$\overline{B_n}$: underestimate of the best probability for n rounds: $\overline{B_n} \leqq B_n$
$T=(T_1, T_2, ..., T_{r-1})$, $T_i=(\alpha_i, \beta_i, \gamma_i, p_i)$, $p_i = xdp(\alpha_i, \beta_i \rightarrow \gamma_i)$, $1 \leqq i < r$
$(\alpha_r, \beta_r, \gamma_r)$: input and output differences to the modular addition at round r
$\widetilde{p_r}$: probability of the partial differential $(\alpha_r[0:i], \beta_r[0:i] \rightarrow \gamma_r[0:i])$
Output - B_n, T: the best probability for n rounds and corresponding trail

1. // Initialization : $r \leftarrow 1, i \leftarrow 0, \alpha_r \leftarrow 0, \beta_r \leftarrow 0, \gamma_r \leftarrow 0$
2. procedure best_diff_search$(r, i, \alpha_r, \beta_r, \gamma_r)$ do
3. //Fisrt round
4. if $(r=1) \wedge (r \neq 1)$ then
5. if $i=w$ then
6. $p_r \leftarrow xdp^+(\alpha_r, \beta_r \rightarrow \gamma_r)$; $T_r \leftarrow (\alpha_r, \beta_r, \gamma_r, p_r)$; add T_r to T
7. $i \leftarrow 0$; $\alpha_{r+1} \leftarrow (\gamma_r \gg r_1)$; $\beta_{r+1} \leftarrow \gamma_r \oplus (\beta_r \ll r_2)$; $\gamma_{r+1} \leftarrow 0$;
8. call best_diff_search$(r+1, i, \alpha_{r+1}, \beta_{r+1}, \gamma_{r+1})$
9. else
10. for $j_\alpha, j_\beta, j_\gamma \in \{0,1\}$ do
11. $\alpha_r[i] \leftarrow j_\alpha$; $\beta_r[i] \leftarrow j_\beta$; $\gamma_r[i] \leftarrow j_\gamma$; $\widetilde{p_r} \leftarrow xdp^+(\alpha_r[0:i], \beta_r[0:i] \rightarrow \gamma_r[0:i])$;
12. if $(\widetilde{p_r} \times B_{n-1}) \geqq \overline{B_n}$ then
13. cal best_diff_search$(r, i+1, \alpha_r, \beta_r, \gamma_r)$
14. //Intermediate rounds
15. if $(r>1) \wedge (r \neq 1)$ then
16. if $i=w$ then
17. $p_r \leftarrow xdp^+(\alpha_r, \beta_r \rightarrow \gamma_r)$; $T_r \leftarrow (\alpha_r, \beta_r, \gamma_r, p_r)$; add T_r to T
18. $i \leftarrow 0$; $\alpha_{r+1} \leftarrow (\gamma_r \gg r_1)$; $\beta_{r+1} \leftarrow \gamma_r \oplus (\beta_r \ll r_2)$; $\gamma_{r+1} \leftarrow 0$;
19. call best_diff_search$(r+1, i, \alpha_{r+1}, \beta_{r+1}, \gamma_{r+1})$
20. else
21. for $j_\gamma \in \{0,1\}$ do
22. $\gamma_r[i] \leftarrow j_\gamma$; $\widetilde{p_r} \leftarrow xdp^+(\alpha_r[0:i], \beta_r[0:i] \rightarrow \gamma_r[0:i])$;
23. if $(\widetilde{p_r} \times B_{n-1}) \geqq \overline{B_n}$ then
24. call best_diff_search$(r, i+1, \alpha_r, \beta_r, \gamma_r)$
25. //Last round
26. if $(r=n)$ then
27. if $i=w$ then
28. $p_r \leftarrow xdp^+(\alpha_r, \beta_r \rightarrow \gamma_r)$; $T_r \leftarrow (\alpha_r, \beta_r, \gamma_r, p_r)$; add T_r to T
29. if $(p_1 \times p_2 ... \times p_n) \geqq \overline{B_n}$ then

30. //Update bound and return to upper round
31. $\overline{B_n} \leftarrow (p_1 \times p_2 \times ... \times p_n)$
32. else
33. for $j_\gamma \in \{0,1\}$ do
34. $\gamma_r[i] \leftarrow j_\gamma$; $\widetilde{p_r} \leftarrow xdp^+(\alpha_r[0:i], \beta_r[0:i] \rightarrow \gamma_r[0:i])$;
35. if $(\widetilde{p_r} \times B_{n-1}) \geqq \overline{B_n}$ then
36. cal best_diff_search$(r, i+1, \alpha_r, \beta_r, \gamma_r)$
37. return

VII. 부록 B

알고리즘 3은 알고리즘 2를 4장에서 설명한 방법으로 수정하여 향상 시킨 것이다. 여기서 형광펜 효과를 준 부분이 알고리즘 2와의 차이가 있는 것이다.

Algorithm 3. Advanced Search for the Best Differential Trail in ARX (Application to SPECK)

Input - n: number of rounds
w: word size in bits
r_1, r_2: right and left rotation constant
r: current round ($n \geqq r \geqq 1$)
i: current bit position ($w > i \geqq 0$)
$B = (B_1, B_2, ..., B_{n-1})$: probabilities of the best trails for rounds 1, 2, ..., (n-1) (global)
$\overline{B_n}$: underestimate of the best probability for n rounds: $\overline{B_n} \leqq B_n$
$T = (T_1, T_2, ..., T_{r-1})$, $T_i = (\alpha_i, \beta_i, \gamma_i, p)$, $1 \leqq i < r$
$(\alpha_r, \beta_r, \gamma_r)$: input and output differences to the modular addition at round r
Output - B_n, T: the best probability for n rounds and corresponding trail

1. // Initialization : $r \leftarrow 1, i \leftarrow 0, \alpha_r \leftarrow 0, \beta_r \leftarrow 0, \gamma_r \leftarrow 0,\ p \leftarrow 1$
2. procedure best_diff_search$(r, i, \alpha_r, \beta_r, \gamma_r, p)$ do
3. //Fisrt round
4. if $(r = 1) \wedge (r \neq 1)$ then
5. if $i = w$ then
6. $p_r \leftarrow xdp^+(\alpha_r, \beta_r \rightarrow \gamma_r)$; $T_r \leftarrow (\alpha_r, \beta_r, \gamma_r, p)$; add T_r to T
7. $i \leftarrow 0$; $\alpha_{r+1} \leftarrow (\gamma_r \ggg r_1)$; $\beta_{r+1} \leftarrow \gamma_r \oplus (\beta_r \lll r_2)$; $\gamma_{r+1} \leftarrow 0$;
8. call best_diff_search$(r+1, i, \alpha_{r+1}, \beta_{r+1}, \gamma_{r+1}, p)$
9. else if $i = 0$
10. for $j_\alpha, j_\beta \in \{0,1\}$
11. $\alpha_r[i] \leftarrow j_\alpha$; $\beta_r[i] \leftarrow j_\beta$; $\gamma_r[i] \leftarrow j_\alpha \oplus j_r$
12. else
13. $temp = (\alpha_r[i-1] = \beta_r[i=1] = \gamma_r[i-1])?\ \alpha_{r[i-1]} : 2$;
14. if $temp = 2$
15. if $(p \times B_{n-1}) \geqq \overline{B_n}$ then
16. for $j_\alpha, j_\beta, j_\gamma \in \{0,1\}$ do
17. $\alpha_r[i] \leftarrow j_\alpha$; $\beta_r[i] \leftarrow j_\beta$; $\gamma_r[i] \leftarrow j_\gamma$; call best_diff_search$(r, i+1, \alpha_r, \beta_r, \gamma_r, p \times 1/2)$
18. else
19. for j_α, j_β $\in \{0,1\}$

20. $\alpha_r[i] \leftarrow j_\alpha$; $\beta_r[i] \leftarrow j_\beta$; $\gamma_r[i] \leftarrow j_\alpha \oplus j_\beta \oplus temp$; call best_diff_search$(r, i+1, \alpha_r, \beta_r, \gamma_r, p)$
21. //Intermediate rounds
22. if $(r > 1) \wedge (r \neq 1)$ then
23. if $i = w$ then
24. $p_r \leftarrow xdp^+(\alpha_r, \beta_r \rightarrow \gamma_r)$; $T_r \leftarrow (\alpha_r, \beta_r, \gamma_r, p)$; add T_r to T
25. $i \leftarrow 0$; $\alpha_{r+1} \leftarrow (\gamma_r \ggg r_1)$; $\beta_{r+1} \leftarrow \gamma_r \oplus (\beta_r \lll r_2)$; $\gamma_{r+1} \leftarrow 0$;
26. call best_diff_search$(r+1, i, \alpha_{r+1}, \beta_{r+1}, \gamma_{r+1}, p)$
27. else if $i = 0$
28. $\gamma_r[0] = \alpha_r[0] \oplus \beta_r[0]$; call best_diff_search$(r, i+1, \alpha_r, \beta_r, \gamma_r, p)$
29. else
30. $temp = (\alpha_r[i-1] = \beta_r[i=1] = \gamma_r[i-1])\ ?\ \alpha_{r[i-1]} : 2$;
31. if $temp = 2$
32. if $(p \times 1/2 \times B_{n-r}) \geqq \overline{B_n}$ then
33. for $j_\gamma \in \{0, 1\}$ do
34. $\gamma_r[i] = j_\gamma$; call best_diff_saerch$(r, i+1, \alpha_r, \beta_r, \gamma_r, p \times 1/2)$
35. else
36. $\gamma_r[i] = \alpha_r[i] \oplus \beta_r[i] \oplus temp$; call best_diff_search$(r, i+1, \alpha_r, \beta_r, \gamma_r, p)$
37. //Last round
38. if $(r = n)$ then
39. if $i = w$ then
40. $p_r \leftarrow xdp^+(\alpha_r, \beta_r \rightarrow \gamma_r)$; $T_r \leftarrow (\alpha_r, \beta_r, \gamma_r, p)$; add T_r to T
41. if $p \geqq \overline{B_n}$ then
42. //Update bound and return to upper round
43. $\overline{B_n} \leftarrow p$
44. else if $i = 0$
45. $\gamma_r[0] = \alpha_r[0] \oplus \beta_r[0]$; call best_diff_search$(r, i+1, \alpha_r, \beta_r, \gamma_r, p)$
46. else
47. $temp = (\alpha_r[i-1] = \beta_r[i=1] = \gamma_r[i-1])\ ?\ \alpha_{r[i-1]} : 2$;
48. if $temp = 2$
49. if $(p \times 1/2 \times B_{n-r}) \geqq \overline{B_n}$ then
50. for $j_\gamma \in \{0, 1\}$ do
51. $\gamma_r[i] = j_\gamma$; call best_diff_saerch$(r, i+1, \alpha_r, \beta_r, \gamma_r, p \times 1/2)$
52. else
53. $\gamma_r[i] = \alpha_r[i] \oplus \beta_r[i] \oplus temp$; call best_diff_search$(r, i+1, \alpha_r, \beta_r, \gamma_r, p)$
54. return

References

[1] Biham, Eli, and Adi Shamir. "Differential cryptanalysis of DES-like cryptosystems," *Journal of CRYPTOLOGY* vol. 4, no. 1, pp. 3-72, Jan. 1991.

[2] Biryukov, Alex, Vesselin Velichkov, and Yann Le Corre. "Automatic search for the best trails in arx: Application to block cipher speck," *Fast Software Encryption - FSE*. pp. 268-288, Mar. 2016.

[3] Beaulieu, Ray, et al. "The SIMON and SPECK lightweight block ciphers," *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015

[4] Lipmaa, Helger, and Shiho Moriai. "Efficient algorithms for computing differential properties of addition," *International Workshop on Fast Software Encryption*. Springer Berlin Heidelberg, 2001

[5] Fu, Kai, et al. "MILP-Based Automatic Search Algorithms for Differential and Linear Trails for Speck," *Fast Software Encryption - FSE*. pp. 289-310, Mar. 2016.

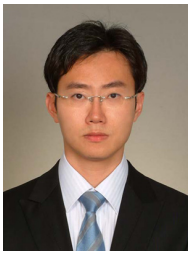
[6] Hong, Deukjo, et al. "HIGHT: A new block cipher suitable for low-resource device," *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer Berlin Heidelberg, 2006

[7] Mouha, Nicky, et al. "Chaskey: an efficient MAC algorithm for 32-bit microcontrollers," *International Workshop on Selected Areas in Cryptography*. Springer International Publishing, 2014

〈저자소개〉

김 서 진 (Seojin Kim) 학생회원
2016년 2월: 고려대학교 수학과 졸업
2016년 3월~현재: 고려대학교 정보보호대학원 석사
〈관심분야〉 대칭키, 공개키 암호, 해쉬함수 설계 및 분석

강 형 철 (Hyngchul Kang) 학생회원
2010년 2월: 고려대학교 산업시스템정보공학과 학사 졸업
2010년 3월~현재: 고려대학교 정보보호대학원 석박사통합과정
〈관심분야〉 블록 암호화 해쉬 함수 설계 및 분석, 인증 암호화 모드 설계



홍 득 조(Deukjo Hong) 종신회원
1999년 8월: 고려대학교 수학과 학사
2001년 8월: 고려대학교 수학과 석사
2006년 2월: 고려대학교 정보보호대학원 박사
2006년 3월~2007년 12월: 고려대학교 정보보호기술연구센터 연구교수
2007년 12월~2015년 8월: 국가보안기술연구소 선임연구원
2015년 9월~현재: 전북대학교 IT정보공학과 조교수
〈관심분야〉 암호 알고리즘 설계 및 분석

성 재 철 (Jaechul Sung) 종신회원
1997년 8월: 고려대학교 수학과 학사
1999년 8월: 고려대학교 수학과 석사
2002년 8월: 고려대학교 수학과 박사
2002년 8월~2004년 1월: 한국정보보호진흥원 선임연구원
2004년 2월~현재: 서울시립대학교 수학과 전임강사, 조교수, 부교수, 교수
〈관심분야〉 암호 알고리즘 설계 및 분석

홍 석 희 (SeokHie Hong) 종신회원
1995년: 고려대학교 수학과 학사
1997년: 고려대학교 수학과 석사
2001년: 고려대학교 수학과 박사
1999년 8월~2004년 2월: ㈜시큐리티 테크놀로지 선임연구원
2003년 3월~2004년 2월: 고려대학교 정보보호기술연구센터 선임연구원
2004년 4월~2005년 2월: K.U. Leuven ESAT/SCD-COSIC 박사후 연구원
2005년 3월~2013년 8월: 고려대학교 정보보호대학원 부교수
2013년 9월~현재: 고려대학교 정보보호대학원 정교수
〈관심분야〉 대칭키 및 공개키 암호 알고리즘, 부채널 공격 및 대응기법, 디지털 포렌식