Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.32, NO.5, Oct. 2022 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2022.32.5.807

Skinny-128-384와 Romulus-N의 SITM 공격*

박 종 현,[1†] 김 종 성[2‡]
[1,2]국민대학교 (대학원생, 교수)

SITM Attacks on Skinny-128-384 and Romulus-N*

Jonghyun Park,[1†] Jongsung Kim[2‡]
[1,2]Kookmin University (Graduate student, Professor)

요 약

SITM (See-In-The-Middle)은 부채널 정보를 차분 분석에 활용하는 분석 기법이다. 이 공격은 블록암호 구현 시 마스킹 되지 않은 중간 라운드의 전력 파형을 수집하여 공격자의 차분 패턴을 만족하는 평문 쌍을 선별하고 이를 차분 분석에 활용하여 키를 복구한다. NIST 경량 암호 표준화 공모사업의 최종 후보 중 하나인 Romulus는 Tweakable 블록암호 Skinny-128-384+를 기반으로 한다. 본 논문에서는 SITM 공격을 14-라운드 부분 마스킹 구현된 Skinny-128-384에 적용하였다. 이 공격은 기 제안된 결과보다 $depth$를 한 라운드 증가한 것뿐만 아니라 시간/데이터 복잡도를 $2^{14.93}/2^{14.93}$으로 줄였다. $Depth$는 전력 파형을 수집하는 블록암호의 라운드 위치를 뜻하며, 이 공격에 대응하기 위해 부분 마스킹 기법 적용 시 필요한 적절한 마스킹 라운드 수를 측정할 수 있다. 더 나아가 공격을 Romulus의 Nonce 기반 AE 모드 Romulus-N으로 확장하였으며, Tweakey의 구조적 특징을 이용하면 Skinny-128-384보다 적은 복잡도로 공격할 수 있음을 보인다.

ABSTRACT

See-In-The-Middle (SITM) is an analysis technique that uses Side-Channel information for differential cryptanalysis. This attack collects unmasked middle-round power traces when implementing block ciphers to select plaintext pairs that satisfy the attacker's differential pattern and utilize them for differential cryptanalysis to recover the key. Romulus, one of the final candidates for the NIST Lightweight Cryptography standardization competition, is based on Tweakable block cipher Skinny-128-384+. In this paper, the SITM attack is applied to Skinny-128-384 implemented with 14-round partial masking. This attack not only increased $depth$ by one round, but also significantly reduced the time/data complexity to $2^{14.93}/2^{14.93}$. $Depth$ refers to the round position of the block cipher that collects the power trace, and it is possible to measure the appropriate number of masking rounds required when applying the masking technique to counter this attack. Furthermore, we extend the attack to Romulus's Nonce-based AE mode Romulus-N, and Tweakey's structural features show that it can attack with less complexity than Skinny-128-384.

Keywords: Differential Cryptanalysis, Side-Channel Analysis, SITM, Skinny-128-384, Romulus-N

Received(08. 05. 2022), Modified(09. 20. 2022), Accepted(09. 21. 2022)
* 본 연구는 고려대 암호기술 특화연구센터(UD210027XD)를 통한 방위사업청과 국방과학연구소의 연구비 지원으로 수행되었습니다.

† 주저자, mmo330@kookmin.ac.kr
‡ 교신저자, jskim@kookmin.ac.kr(Corresponding author)

I. 서 론

과학기술의 발전으로 사물 인터넷(IoT) 기기의 수요는 지속해서 증가하고 있다. 특히, 사용자의 편의성을 고려하여 가볍고 작게 설계된 사물 인터넷 기기는 사용 가능한 자원이 한정된다. 경량 암호는 이와 같은 사물 인터넷 환경에 사용하기 적합하다.

미국 국립표준기술연구소(NIST)에서는 경량 암호 표준화 공모사업이 진행되고 있다. 공모사업의 최종 후보에는 Tweakable 블록암호 Skinny-128-384+를 기반으로 하는 Romulus[1]가 포함되어 있다.

수학을 기반으로 하는 대표적인 암호분석 기법에는 차분 분석(Differential Cryptanalysis, DC)이 있다[2]. 이 공격은 특정 비트가 반전된 평문 쌍을 암호화하여 생성된 암호문 쌍을 이용해 키를 복구한다.

물리적으로 발생한 부가적인 정보를 이용하는 부채널 분석(SCA)은 암호화 연산 시 발생한 소요 시간, 전자파, 전력 소모량 등을 획득하여 이를 이용해 암호를 분석하는 기법이다[3]. 부채널 분석에 대응하는 방법에는 마스킹 기법[4]이 있다.

DATE 2018에는 부채널 정보를 차분 분석에 활용하는 분석 기법인 SCADPA (Side-Channel Assisted Differential Plaintext Attack)가 제안되었다[5]. 이 분석 기법은 공격의 대상이 Bit-permutation 기반 블록암호로 한정되었지만, TCHES 2020에 제안된 SITM (See-In-The-Middle)은 SPN 구조의 블록암호로 대상이 확장되었다[6]. 이 분석 기법은 블록암호 구현 시 마스킹 기법이 적용되지 않은 라운드의 전력 파형 정보를 수집한다. 이 전력 파형은 서로 다른 평문 쌍 암호화 시 공격자가 예측한 차분 경로를 만족하였는지를 판단하는 데 사용된다. 이를 통해 평문 쌍을 선별하고 차분 분석에 활용한다. 이 공격을 이용하면 블록암호 구현 시 부분 1차·고차 마스킹 기법을 적용하는 적절한 라운드 수를 판단할 수 있고 이는 마스킹 기법으로 발생하는 비용을 줄일 수 있다.

본 논문에서는 블록암호 Skinny-128[7]의 차분 경로를 소개하고 이를 통해 LUT (Look Up Table)로 구현된 Skinny-128-384의 향상된 SITM 공격을 제시한다. 최종적으로는 제안하는 공격을 Romulus의 Nonce 기반 AE 모드 Romulus-N에 적용하는 방안을 소개한다.

본 논문의 구성은 다음과 같다. 2장은 Skinny-128-384 및 Romulus-N의 알고리즘과 SITM 개요를 설명한다. 3장에서는 SITM 공격에 사용하는 Skinny-128의 차분 경로를 소개하고 향상된 Skinny-128-384의 SITM 공격을 설명한다. 4장에서는 Romulus-N에 SITM 공격을 적용해 키를 복구하는 방안을 제시한다. 마지막으로 5장은 결론 및 향후의 계획으로 마무리한다.

II. 배경지식

2.1 Skinny-128-384 알고리즘

CRYPTO 2016에 제안된 Skinny-128-384는 블록 크기 128-비트와 Tweakey 크기 384-비트를 사용한다. Skinny-128-384의 알고리즘 설명에는 다음과 같은 표기를 사용한다.

암호의 State는 Fig. 1.과 같이 표현하자.

Skinny-128-384는 전체 56-라운드이며, 화이트닝 키를 사용하지 않는다. 라운드 함수는 다음과 같다.

- SubCells: 전체 State에 바이트 단위로 S-box가 적용되는 함수, S-box: $\{0,1\}^8 \rightarrow \{0,1\}^8$
- AddConstants: State의 0, 4, 8번째 바이트에 고정된 상수가 XOR 되는 함수, 이때 사용된 상수를 차례로 c_0, c_1, c_2라고 하자.
- AddRoundTweakey: State의 상위 64-비트를 라운드키로 XOR 하는 함수, 이때, 사용되는 라운드키의 State는 Fig. 2.와 같다.

$$\begin{bmatrix} m_0 & m_1 & m_2 & m_3 \\ m_4 & m_5 & m_6 & m_7 \\ m_8 & m_9 & m_{10} & m_{11} \\ m_{12} & m_{13} & m_{14} & m_{15} \end{bmatrix}$$

Fig. 1. Cipher State with labeled bytes.

Table 1. Notations in Skinny-128-384.

Notations	Description
m_i	i^{th} byte in State
rk_i	i^{th} byte in Roundkey ($i=0,\ldots,7$)
tk_i	i^{th} byte in Tweakey ($i=0,\ldots,47$)
$TK1_i$	$TK1_i = tk_i$ ($i=0,\ldots,15$)
$TK2_i$	$TK2_i = tk_{16+i}$ ($i=0,\ldots,15$)
$TK3_i$	$TK3_i = tk_{32+i}$ ($i=0,\ldots,15$)
$\mid x \mid$	Bit length of x
S_i	State after the i^{th}-Round SubCells ($i=1,\ldots,56$)

$$\begin{bmatrix} rk_0 & rk_1 & rk_2 & rk_3 \\ rk_4 & rk_5 & rk_6 & rk_7 \\ - & - & - & - \\ - & - & - & - \end{bmatrix}$$

Fig. 2. Roundkey State with labeled bytes.

- ShiftRows: State의 각 행을 바이트 단위로 회전하는 함수(r행은 오른쪽으로 r번 회전, r=0, 1, 2, 3)
- MixColumns: State의 행 단위로 XOR 되는 함수, 동작 과정은 라운드 함수인 Fig.3.을 통해 알 수 있음.

AddRoundTweaky에 사용되는 초기 라운드키는 다음과 같이 생성된다.

- $rk_i = TK1_i \oplus TK2_i \oplus TK3_i,\ i = 0, \cdots, 7.$

$TK1$, $TK2$, $TK3$은 Tweakey schedule을 통해 각각 갱신되며, 사용되는 함수들은 아래와 같다.

- P_T: 전체 16개 바이트 위치를 치환하는 함수, 자세한 과정은 다음과 같음.

- $(0, 1, \ldots, 15) \xrightarrow{P_T} (9, 15, 8, 13, 10, 14, 12, 11, 0, 1, \ldots, 7).$

- LFSR: $TK1$, $TK2$, $TK3$의 각 바이트를 비트 단위의 치환 및 XOR 하는 함수, $TK2$와 $TK3$에서 사용되는 LFSR은 서로 다르며, 상위 8-바이트에만 적용됨.

- LFSR (Case $TK2$):
  $(x_7 \| x_6 \| \ldots \| x_1 \| x_0)$
  $\rightarrow (x_6 \| x_5 \| \ldots \| x_0 \| x_7 \oplus x_5)$,

- LFSR (Case $TK3$):
  $(x_7 \| x_6 \| \ldots \| x_1 \| x_0)$
  $\rightarrow (x_0 \oplus x_6 \| x_7 \| x_6 \| \ldots \| x_1)$
  이때, x_0는 LSB (Least Significant Bit)임.

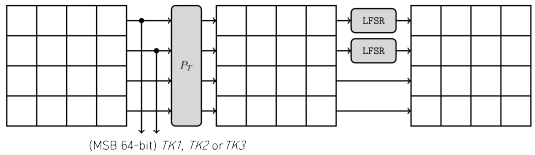

Fig. 4. Tweakey schedule in Skinny-128-384 (MSB 64-bit is leftmost 64-bit of State).

Tweakey schedule은 P_T와 LFSR을 사용하여 Fig. 4.와 같이 동작한다. $TK1$은 LFSR이 적용되지 않는다.

Tweakey schedule을 통해 갱신된 $TK1$, $TK2$, $TK3$의 상위 64-비트는 다음 라운드키 생성에 사용된다.

2.2 Romulus-N 알고리즘

Romulus-N은 Romulus의 Nonce 기반 AE 모드이다. 핵심 함수로 Skinny-128-384+를 사용하며, 이는 기존의 전체 56-라운드에서 40-라운드로 축소 시킨 차이가 있다. Romulus-N의 설명에는 다음과 같은 표기를 사용한다.

Romulus-N은 인코딩을 통해 384-비트 Tweakey를 생성한다. 인코딩의 입력 파라미터는 마스터키 K, 128-비트 임시변수 Te, 특정 규칙으로 정해지는 상수 D와 B가 있다.

Table 2. Notations in Romulus-N

Notations	Description
N	Nonce
$A[i]$	i^{th} block in Associated data ($i = 1, \ldots, a$)
$M[i]$, $C[i]$	i^{th} block in Message, Ciphertext ($i = 1, \ldots, m$)
$E_{(K, Te, B, D)}$	Block cipher encryption using encoded Tweakey
i^j	Bit where i is connected j times, $i \in \{0, 1\}$, $j \in \{1, 2, 3, \ldots\}$
T	Tag

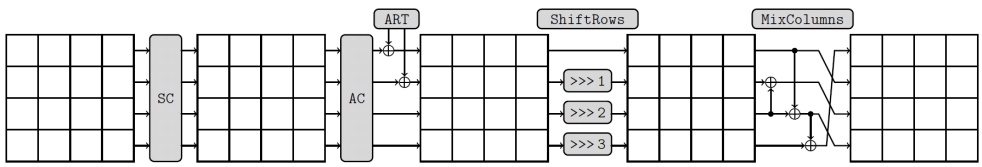

Fig. 3. The Skinny-128-384 round function applies five different transformations: SubCells (SC), AddConstants (AC), AddRoundTweakey (ART), ShiftRows (SR), MixColumns (MC).

$D(=z_{55}\,||\,\ldots\,||\,z_0)$는 Romulus-N의 암호화된 연관 데이터 또는 메시지 블록의 개수이다. D는 lfsr_{56}을 통해 변환되며, 그 과정은 다음과 같다.

- lfsr_{56}:
  if $i=7$, $z_7 \leftarrow z_6 \oplus z_{55}$.
  if $i=4$, $z_4 \leftarrow z_6 \oplus z_{55}$.
  if $i=2$, $z_2 \leftarrow z_1 \oplus z_{55}$.
  if $i=7$, $z_7 \leftarrow z_{55}$.
  else, $z_i \leftarrow z_{i-1}$.

$B(=b_7\,||\,b_6\,||\,\ldots\,||\,b_0)$는 사용되는 Romulus의 모드, 암호화되는 대상(연관 데이터 또는 메시지) 및 블록 위치 그리고 패딩의 필요성에 따라 값이 정해진다. Romulus-N을 기준으로 B는 다음과 같이 정해진다. 이때, 연관 데이터의 블록 개수는 a이고 메시지 블록의 개수는 m이다.

1) $b_7 b_6 b_5 = 000$.
2) 마지막 블록의 암호화라면 $b_4=1$, 아니면 $b_4=0$.
3) 연관 데이터 암호화 구간이라면 $b_3=1$, 아니면 $b_3=0$.
4) 메시지 암호화 구간이라면 $b_2=1$, 아니면 $b_2=0$.
5) $A[a]$에 패딩이 필요하면 $b_1=1$, 아니면 $b_1=0$.
6) $M[m]$에 패딩이 필요하면 $b_0=1$, 아니면 $b_0=0$.

Te는 연관 데이터의 블록 또는 Nonce가 사용된다. $encode$의 자세한 과정은 다음과 같다.

- $encode(K, Te, B, D) = \mathrm{lfsr}_{56}(D)\,||\,B\,||\,0^{64}\,||\,Te\,||\,K$.

Romulus-N에서 사용되는 함수들은 아래와 같다.

- pad: 메시지 및 연관 데이터의 마지막 블록에 패딩을 적용하는 함수, 자세한 과정은 다음과 같음.

- $pad(X)$:
  if $|X|=128$, $pad(X)=X$,
  else, $pad(X)=X\,||\,0^{128-|X|-8}\,||\,len_8(X)$.
  이때, $len_8(X)$는 X의 바이트 길이를 8-비트 크기 값으로 나타내는 함수임.

- G: 바이트 단위의 8×8 binary matrix를 행렬곱하는 함수, 그 과정은 아래와 같음.

- $G(X)=(G_s \cdot X[0],\ G_s \cdot X[1],\ \ldots,\ G_s \cdot X[15])$.
  이때 8×8 binary matrix G_s는 Fig. 6.과 같으며, 바이트 크기 $X[i]$를 행렬 곱 연산함.

- ρ: G의 출력값과 메시지로 암호문을 생성하고 G의 입력값과 메시지로 $E_{(K,\,Te,\,B,\,D)}$의 입력값을 생성하는 함수, 자세한 과정은 다음과 같음.

- $\rho(S, M) = (S', C) = (S \oplus M,\ M \oplus G(S))$

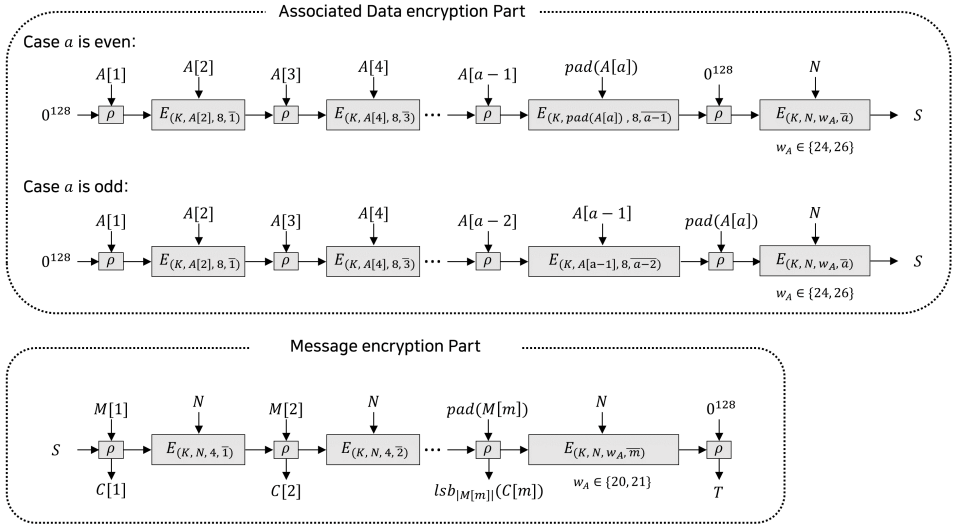

Fig. 5. The Romulus-N Nonce-based AE mode. $\mathrm{lfsr}_{56}(D)$ is $\overline{D}$ and $lsb_a(X)$ is the rightmost a-bit of X.

Romulus-N의 메시지 암호화 및 태그 생성 과정은 Fig. 5와 같다.

2.3 SITM 개요

SITM 공격은 암호화 과정 중 발생한 전력 파형의 차이를 이용하여 평문 쌍을 선별 및 수집하고 이를 차분 분석에 활용한다. 설명에는 다음과 같은 정의들이 사용된다.

- 전력 파형: 암호 알고리즘이 탑재된 기기로부터 발생하는 전력의 흐름 그래프
- 차분 파형: 두 전력 파형의 차이
- 차분 경로: 두 평문 쌍이 암호화되면서 전파되는 차분 예상 경로
- 차분 패턴: 여러 차분 경로들을 한 개의 패턴으로 나타낸 경로
- Active S-box: 차분 경로 내 S-box의 입력 차분이 0이 아닌 경우
- Inactive S-box: 차분 경로 내 S-box의 입력 차분이 0인 경우
- $Depth$: 전력 파형을 수집하는 블록암호의 라운드 위치, (e.g. $Depth=8$이면, 암·복호화를 모두 고려하여 대상은 전체 라운드 중에서 14-라운드만 마스킹 기법을 적용함)

SITM은 LUT 구현 기반의 전체 라운드 블록암호를 공격 대상으로 하며, 공격 과정은 **평문 선별 알고리즘**과 **키 복구 알고리즘**으로 나뉜다.

평문 선별 알고리즘은 암호화 과정 중 발생하는 전력 파형의 차이를 활용한다. 서로 다른 평문 쌍이 암호화되는 과정에서 발생한 전력 파형을 수집하였다고 가정하자. 만약 같은 위치의 S-box 연산 시 입력값이 서로 일치하였다면 이때 발생한 두 전력 파형은 유사할 것이고, 그렇지 않다면 두 전력 파형은 상이할 것으로 예측할 수 있다. 이 논리를 이용해 공격자가 예상한 차분 경로를 만족하는 평문 쌍을 선별할 수 있다.

키 복구 알고리즘은 선별한 평문과 차분 특성을 활용하여 키 후보를 구한다. 키 복구 과정은 블록암호의 차분 특성, 차분 패턴 그리고 생성된 키 후보의 개수에 따라 달라진다. 따라서 자세한 설명은 3장에서 소개한다.

$$G_s = \begin{pmatrix} 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1\\ 1&0&0&0&0&0&0&1 \end{pmatrix}$$

Fig. 6. 8×8 binary matrix G_s

III. 향상된 Skinny-128-384 SITM 공격

3.1 기 제안된 7-라운드 Skinny-128 차분 패턴

기 제안된 Skinny-128의 SITM 공격[6]은 7-라운드 Skinny-128 차분 패턴을 사용하였다. 해당 패턴은 S_1의 3, 6, 9번째 바이트의 차분과 4, 11, 14번째 바이트의 차분이 일치한다. 해당 7-라운드 차분 패턴은 Fig. 7.과 같다.

Skinny는 화이트닝 키를 사용하지 않기에, 공격자는 임의로 S_1 위치의 차분을 설정할 수 있다. 해당 공격은 S_2의 7, 10, 13번째 바이트의 차분이 서로 일치하게 되면서 S_3의 Active S-box 개수가 1개가 되는 확률을 2^{-22}으로 계산하였다. S_1의 0이 아닌 차분은 고정된 값이 아니다.

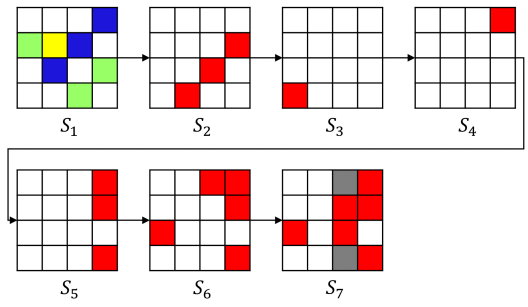

Fig. 7. A 7-round Skinny-128 differential pattern. Active bytes are colored in red. blue, green. Each blue and green bytes have the same difference value. Grey bytes may or may not be active.

3.2 8-라운드 Skinny-128 차분 패턴

본 논문에서는 위 7-라운드 차분 패턴을 기반으로 한 라운드 연장하는 것뿐만 아닌 확률을 약 $2^{-4.79}$으로 높인 차분 패턴들을 제안한다. 제안하는 8-라운드 Skinny-128 차분 패턴 1~4는 Fig. 8.과 같다.
제안한 차분 패턴의 S_1에서 0이 아닌 차분은 모두 같으며, 이 값을 u라고 하자. u의 선택에 따라 차분

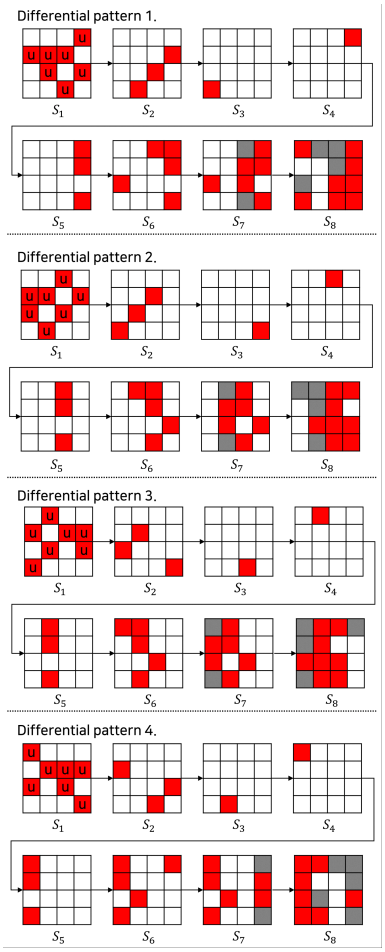

Fig. 8. Four 8-round Skinny-128 differential patterns. Active bytes are colored in red.

패턴의 확률이 달라지며, 높은 확률을 갖는 u를 찾기 위해 조사하였고 그 결과를 Table 3.에 제시한다.

차분 패턴의 확률이 가장 높은 상위 6개의 u는 2, 32, 16, 48, 128, 144가 있다. 이를 이용하면, 제안한 차분 패턴마다 6개의 8-라운드 차분 패턴을 생성할 수 있고 이때의 평균 확률은 $2^{-4.79}$이다. 기존의 차분 패턴은 S_1에서 0이 아닌 차분으로 최대 3가지 종류의 값이 생성될 수 있다. 하지만 S-box의 입력 차분에 따라서 가능한 출력 차분이 다르기 때문에, 기존 차분 패턴은 불가능한 차분 특성이 발생한다.

Table 3. Probability of six 8-round Skinny-128 differential patterns using u.

u	2	32	16	48	128	144
Pr.	$2^{-4.68}$	$2^{-4.68}$	$2^{-4.79}$	$2^{-4.79}$	$2^{-4.88}$	$2^{-4.93}$

3.3 향상된 Skinny-128-384에 대한 SITM 공격

기 제안된 SITM 공격은 $Depth=7$로 설정한 12-라운드 부분 마스킹 기법이 적용된 Skinny-128-384를 공격하였다. 본 절에서는 $Depth=8$로 설정한 14-라운드 부분 마스킹 기법이 적용된 Skinny-128-384를 공격한다.

공격 과정은 **평문 선별 알고리즘**과 **키 복구 알고리즘**으로 나뉜다. 평문 선별 알고리즘은 제안한 8-라운드 차분 패턴 1~4를 각각 사용하며, 차분 패턴마다 6개의 u를 모두 사용한다. 키 복구 알고리즘은 선별한 평문과 차분 패턴의 특성을 이용한다. 본 공격에서는 PRESENT의 SITM 공격 논문[8]에서 사용한 키를 복구하는 논리를 사용한다.

평문 선별 알고리즘은 다음과 같다.

아래의 과정은 차분 패턴 1을 기준으로 설명한다.

1) 차분 패턴의 S_1을 만족하는 평문 쌍 1개를 랜덤하게 생성한다.

2) 평문 쌍을 암호화하여 8-라운드의 SubCells 동작 과정에서 발생한 전력 파형을 수집한다.

3) 수집한 두 쌍의 전력 파형으로 차분 파형을 계산한다.

4) 차분 파형 관찰한다. 이때, 8-라운드에서 4, 5, 9, 13번째 S-box에 대한 차분 파형만 현저히 낮으면 inactive S-box로 판단하고 사용된 평문을 선별한다. 그렇지 않다면 1) 과정으로 돌아가 다른 평문 쌍을 생성하여 만족할 때까지 반복한다.

키 복구 알고리즘은 선별된 평문과 차분 패턴의 특성을 이용하기에 다음과 같은 설명들이 필요하다. Fig. 9는 차분 패턴 1의 앞 2-라운드 상세 경로이다.

Fig. 9.의 차분 패턴에서 차분 v는 입력 차분 u가 정해진 값이기에, DDT를 통해 가능한 v의 후보들을 유추할 수 있다. S-box의 입력 차분과 출력

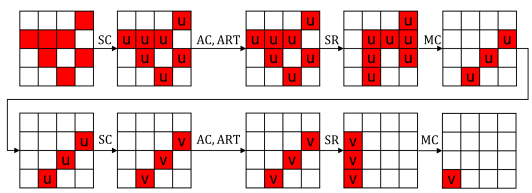

Fig. 9. Forward 2-round differential pattern in differential pattern 1.

Table 4. Number of input value list in Active S-box before 2-round SubCells

u	2	32	16	48	128	144
N.	$2^{19.32}$	$2^{19.32}$	$2^{19.21}$	$2^{19.21}$	$2^{19.12}$	$2^{19.12}$

차분이 고정되면 이때 사용된 입력값 후보를 알 수 있다. 이를 이용하면 2-라운드 SubCells 이전 State의 7, 10, 13번째 바이트의 S-box 입력값 후보를 계산할 수 있다. u 선택에 따른 입력값 후보들의 개수는 DDT를 이용하면 계산할 수 있으며, 그 결과를 Table 4.에 제시한다.

1-라운드 SubCells 이후의 바이트를 p_i', 2-라운드 SubCells 이전의 바이트를 s_i라고 표현하자. s_i는 다음과 같은 수식으로 나타낼 수 있다. 이때, 파란색으로 표기된 변수는 공격자가 알 수 있는 정보이다.

- $s_0 = p_0' \oplus p_{10}' \oplus p_{13}' \oplus c_0 \oplus rk_0$
- $s_1 = p_1' \oplus p_{11}' \oplus p_{14}' \oplus rk_1$
- $s_2 = p_2' \oplus p_8' \oplus p_{15}' \oplus c_2 \oplus rk_2$
- $s_3 = p_3' \oplus p_9' \oplus p_{12}' \oplus rk_3$
- $s_4 = p_0' \oplus c_0 \oplus rk_0$
- $s_5 = p_1' \oplus rk_1$
- $s_6 = p_2' \oplus rk_2$
- $s_7 = p_3' \oplus rk_3$
- $s_8 = p_7' \oplus p_{10}' \oplus rk_7$
- $s_9 = p_4' \oplus p_{11}' \oplus c_1 \oplus rk_4$
- $s_{10} = p_5' \oplus p_8' \oplus c_2 \oplus rk_5$
- $s_{11} = p_6' \oplus p_9' \oplus rk_6$
- $s_{12} = p_0' \oplus p_{10}' \oplus c_0 \oplus rk_0$
- $s_{13} = p_1' \oplus p_{11}' \oplus rk_1$
- $s_{14} = p_2' \oplus p_8' \oplus c_2 \oplus rk_2$
- $s_{15} = p_3' \oplus p_9' \oplus rk_3$

키 복구 알고리즘은 다음과 같다.

1) 2-라운드 SubCells 이전 State의 7, 10, 13번째 입력값 후보들과 위 수식을 이용하면 rk_1, rk_3, rk_5 후보들을 계산할 수 있다. 이 후보들의 개수는 입력값 후보들의 개수만큼 생성된다.

2) 생성된 rk_1, rk_3, rk_5 후보들을 저장한다.

rk_1, rk_3, rk_5 후보에서 잘못된 키를 소거하는 방법은 [8]의 논리를 이용하였다. 6개의 u를 모두 사용하기 위해 **평문 선별 알고리즘**과 **키 복구 알고리즘**을 반복하면 6개의 키 후보 집합 얻을 수 있다. 각 집합의 원소 개수는 Table 4.를 통해 알 수 있다. 3-바이트 크기 전체 집합의 부분 집합인 키 후보 집합 6개를 교집합 하면 아래와 같이 잘못된 키가 남아 있을 확률을 계산할 수 있다.

- $2^{24-4.68-4.79-4.68-4.79-4.88-4.88} = 2^{-4.7} (< 1)$

나머지 1-라운드 바이트는 차분 패턴 2~4를 통해 구할 수 있다. 다음 라운드키는 1-라운드키를 이용하여 앞 1-라운드를 연산할 수 있기에, $depth$를 1 증가하여 위 공격과 유사한 방법으로 구할 수 있다. [6]에서 제안한 키 복구 논리는 생성된 키 후보 집합의 원소 개수만큼 추가적인 공격 복잡도 발생하기 때문에, [8]의 논리를 사용하는 것이 더 적은 공격 복잡도를 유도할 수 있다.

Tweakey 384-비트를 복구하기 위해서는 1~6-라운드키가 필요하다[6].

3.4 Skinny-128-384의 SITM 공격 실험

본 논문에서는 전력 파형을 수집 및 관찰하는 장비가 있다고 가정하고 1-라운드키를 복구하는 공격을 실험하였다. 본 공격은 잘못된 키 후보를 줄이기 위한 키 후보 집합의 적절한 개수를 판단하기 위해 u를 6개부터 17개까지 1개씩 증가하면서 실험하였다. 사용된 u는 다음과 같다.

- u = 2, 32, 16, 48, 128, 144, 1, 33, 64, 80, 192, 208, 8, 9, 5, 4, 10.

위 u의 순서는 차분 패턴의 확률이 높은 순서로 되어 있으며, 확률이 높은 6개부터 17개까지 순차적으로 500회의 실험을 진행하였다. 이를 통해, 평균적인 데이터 개수와 복구된 라운드키 후보의 개수 및 복구 성공률을 구하였고 그 결과를 Table 5.에 제시한다.

잘못된 키를 소거하기 위해 확률적으로 필요한 u의 개수를 6개로 계산하였지만, 위 실험 결과를 통해 9개 이상의 u를 사용해야만 평균적으로 라운드키 후보의 개수가 약 1개가 되는 것을 알 수 있다. 1-

Table 5. Experiment results of SITM attacks on Skinny-128-384 using 8-round differential patterns. #(u) is the number of u used in attack and #(R) is the number of 1-Roundkey Candidates.

#(u)	6	7	8	9	10	11
Data	$2^{9.39}$	$2^{9.72}$	$2^{9.98}$	$2^{10.21}$	$2^{10.42}$	$2^{10.61}$
#(R)	$2^{12.56}$	$2^{4.54}$	$2^{2.23}$	1.72	1.4	1.26
Pr.	1	1	1	1	1	1
#(u)	12	13	14	15	16	17
Data	$2^{10.76}$	$2^{10.89}$	$2^{11.03}$	$2^{11.16}$	$2^{11.27}$	$2^{11.37}$
#(R)	1.29	1.26	1.25	1.06	1.02	1.02
Pr.	1	1	1	1	1	1

라운드키는 다음 라운드키를 복구하는 과정에서 사용되기에, 라운드키 후보의 개수가 적을수록 공격 복잡도를 줄일 수 있다. 따라서, 16개의 u를 사용하는 것을 기준으로 공격 복잡도를 계산하면 다음과 같다.

- Time Complexity:
$2\times(2^{11.27}+1.02\times 2^{11.27}+1.02^2\times 2^{11.27}+...+1.02^5\times 2^{11.27})$
$\approx 2^{14.93}$

- Data Complexity:
$2\times(2^{11.27}+1.02\times 2^{11.27}+1.02^2\times 2^{11.27}+...+1.02^5\times 2^{11.27})$
$\approx 2^{14.93}$

- Memory Complexity:
$2^{19.32}+2^{19.32}=2^{20.32}$

시간 및 데이터 복잡도는 패턴 1~4에 대해 u를 16번 변경하면서 평문 선별 알고리즘을 사용하는 과정과 이 과정을 1~6-라운드키를 구하기 위해 6번 반복하는 복잡도를 고려하였다.

공간 복잡도는 교집합 연산에 사용되는 2개의 키 후보 집합을 저장하는데 필요한 최대 바이트 수이다.

IV. Romulus-N SITM 공격

4.1 Romulus-N의 마스터키 복구 논리

Romulus-N의 마스터키 K는 Skinny-128-384+의 1~2-라운드키를 통해 복구할 수 있다. Tweakey의 상위 256-비트는 알려진 상수와 연관 데이터 또는 Nonce로 구성되어 있다. 이를 이용하면 1-라운드키를 통해 K의 상위 64-비트를 복구할 수 있다. 그 과정은

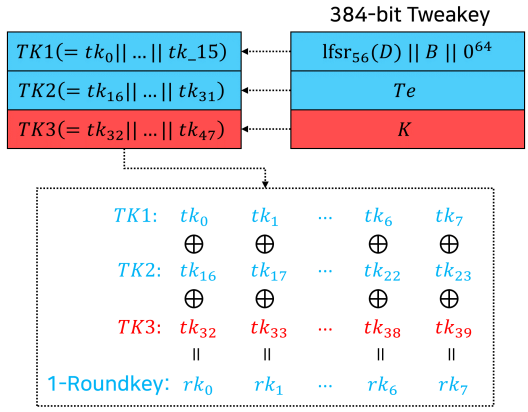

Fig. 10. Leftmost 64-bit masterkey recovery using 1-Roundkey. Blue value is known information and red is unknown information.

Fig. 10.과 같다.

K의 하위 64-비트는 Tweakey Schedule로 갱신된 $TK1$, $TK2$, $TK3$과 2-라운드키를 이용하여 위와 같은 논리를 통해 복구할 수 있다. 갱신된 $TK1$, $TK2$, $TK3$의 구조는 Fig. 11.과 같으며. 상단부터 $TK1$, $TK2$, $TK3$이다.

갱신된 $TK3$에서 선형함수 LFSR이 적용된 바이트를 역연산하면 K의 하위 64-비트를 복구할 수 있다.

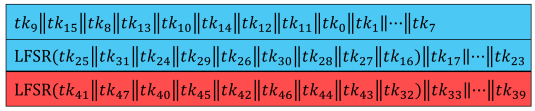

Fig. 11. $TK1$, $TK2$ and $TK3$ value after Tweakey Schedule. Blue is known information and red is unknown information.

4.2 Romulus-N에 대한 SITM 공격

SITM 공격으로 Romulus-N의 모든 블록에서 사용되는 Tweakey를 복구하는 공격 가정은 다음과 같다.

- Nonce 및 연관 데이터는 알려진 정보
- 연관 데이터가 2개 블록 이상 필요

Romulus-N에 SITM 공격을 적용하는 위치는 Fig. 12.를 통해 알 수 있다.

공격이 적용되는 블록의 암호화 대상은 $A[1]$과 0^{128}을 XOR 연산한 값이기에, $A[1]$이 그대로 암호화된다. 공격자는 $A[1]$을 조작하여 3장에서 제안한 SITM 공격을 적용한다. Romulus-N은 알려진 정보인 Nonce와

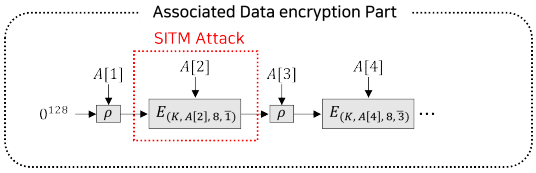

Fig. 12. Skinny-128-384+ SITM Attack area on Romulus-N.

연관 데이터를 Tweakey에 사용하기에, K를 구하면 모든 블록에서 사용하는 Tweakey를 복구할 수 있다. 공격 복잡도는 3.4절에서 제시한 공격 복잡도 계산에서 1~2-라운드키를 구하는 차이가 있다. Skinny-128-384와 Romulus-N에 대한 공격 복잡도를 정리하면 Table 6.과 같다.

Table 6. Comparison of the SITM attack complexities on Skinny-128-384 and Romulus-N.

Cipher	*Depth*	Data	Time	Memory	Ref.
Skinny-128-384	7	$2^{26.58}$	$O(2^{22})$	$2^{19.59}$	[6]
	8	$2^{14.93}$	$2^{14.93}$	$2^{20.32}$	Ours
Romulus-N	8	$2^{13.28}$	$2^{13.28}$	$2^{20.32}$	Ours

V. 결 론

SITM은 암호화 과정 중 발생한 전력 파형의 차이를 통해 얻은 정보를 차분 분석에 활용하는 공격이다. 본 논문에서는 SITM 공격을 14-라운드 부분 마스킹 기법이 적용된 Skinny-128-384에 적용하였다. 그 결과 기 제안된 공격보다 한 라운드 높은 *depth*로 공격이 가능함을 보였고 데이터/시간 복잡도를 줄였다. 더 나아가 공격을 Romulus-N으로 확장하였으며, Tweakey의 구조적 특징을 이용해 더 적은 라운드키로 사용되는 전체 Tweakey를 복구할 수 있음을 보였다.

해당 공격에 대응하기 위해서는 Skinny-128-384 구현 시 적어도 16-라운드 부분 마스킹 기법 적용해야 한다. 제안된 모든 SITM 공격들은 LUT로 구현된 블록암호를 대상으로 하고 있다. 이는 분명한 한계점이기에, 비트 슬라이스로 구현된 블록암호를 공격할 수 있는 전용 차분 경로 및 키 복구 논리를 향후 연구할 계획이다.

References

[1] Romulus, https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/romulus-spec-final.pdf

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

[3] P. Kocher, J. Jaffe and B. Jun, "Differential power analysis," In: Annual international cryptology conference, LNCS 1666, pp. 388-397, Dec. 1999.

[4] S. Nikova, C. Rechberger, and V. Rijmen, "Threshold Implementations Against Side-Channel Attacks and Glitches," International conference on information and communications security, LNCS 4307, no. 1, pp. 529 - 545, Dec. 2006.

[5] J. Breier, D. Jap, and S. Bhasin, "SCADPA: Side-channel assisted differential-plaintext attack on bit permutation based ciphers," 2018 Design, Automation & Test in Europe Conference & Exhibition, IEEE, pp. 1129-1134, Mar. 2018.

[6] S. Bhasin, J. Breier, X. Hou, D. Jap, R. Poussier and S. M. Sim, "Sitm: See-in-the-middle side-channel assisted middle round differential cryptanalysis on spn block ciphers," IACR Transactions on Cryptographic Hardware and Embedded Systems, vol. 2020, no. 1, pp. 95-122, Nov. 2019.

[7] C. Beierle, et al. "The SKINNY family of block ciphers and its low-latency variant MANTIS," In Annual International Cryptology Conference, pp. 123-153, Aug. 2016

[8] J.H Park, H.G Kim, and J.S Kim, "Improved SITM Attack on the PRESENT Block cipher," Journal of the Korea Institute of

Information Security & Cryptology, 32(2),
pp. 155-162, Apr. 2022.

〈저자소개〉

박 종 현 (Jonghyun Park) 학생회원
2021년 2월: 국민대학교 정보보안암호수학과 졸업
2021년 3월~현재: 국민대학교 금융정보보안학과 석사과정
〈관심분야〉 정보보호, 암호 알고리즘

김 종 성 (Jongsung Kim) 종신회원
2006년 11월: K.U.Leuven, ESAT/COSIC 정보보호 전공 공학박사
2007년 2월: 고려대학교 정보보호대학원 공학박사
2009년 9월~2013년 2월: 경남대학교 e-비즈니스학과 교수
2013년 3월~2017년 2월: 국민대학교 수학과 교수
2017년 3월~현재: 국민대학교 정보보안암호수학과/금융정보보안학과 교수
〈관심분야〉 정보보호, 암호 알고리즘, 디지털 포렌식