2byte mask 1차 마스킹 PIPO 제안

https://youtu.be/scrhC6cNT9k

기존연구 : 2개의 비트를 사용하는 1차 마스킹

Gross et al. First-Order Masking with Only Two Random Bits

- 마스킹은 오래전부터 부채널 공격에 효율적인 대응기법
  그러나 마스킹에 필요한 렌덤 비트 생성은 많은 비용이 듬

➢ 마스킹을 최소화 하는 것을 목표

➢ 2 비트를 사용하는 1차 마스킹 기법 제안

➢ 새로운 마스킹 AND 게이트 제안

➢ 2 비트의 무작위성으로 AES구현하고 보안 분석

masked XOR

마스킹 목표 : 민감한 정보를 무작위 데이터와 결합하여 보호

a0 = a ⊕ m0, a1 = m0

b0 = b ⊕ m1, b1 = m1

q0 = a0 ⊕ b0

q1 = a1 ⊕ b1

q = q0 ⊕ q1

= (a ⊕ m0) ⊕ (b ⊕ m1) ⊕ m0 ⊕ m1 = a ⊕ b

Table 1: Truth table of the masked XOR from Eqn. 1

Shares				*Secrets*			*TT*	
a_0	a_1	b_0	b_1	a	b	$a \oplus b$	q_0	q_1
0	0	0	0	**0**	**0**	**0**	0	0
0	0	1	1				1	1
1	1	0	0				1	1
1	1	1	1				0	0
TT Hamming Weight:							**2**	**2**
0	0	0	1	**0**	**1**	**1**	0	1
0	0	1	0				1	0
1	1	0	1				1	0
1	1	1	0				0	1
TT Hamming Weight:							**2**	**2**
0	1	0	0	**1**	**0**	**1**	0	0
0	1	1	1				1	1
1	0	0	0				0	0
1	0	1	1				1	1
TT Hamming Weight:							**2**	**2**
0	1	0	1	**1**	**1**	**0**	0	1
0	1	1	0				1	0
1	0	0	1				0	1
1	0	1	0				1	0
TT Hamming Weight:							**2**	**2**

masked XOR

동일한 마스크 사용은 위험

$q0 = a0 \oplus b0 = (a \oplus m0) \oplus (b \oplus m0) = a \oplus b$

$a \oplus b$가 노출

해결 : 여러 마스킹 사용

$q0 = a0 \oplus b0 \oplus c0 \oplus \cdots \oplus z0$
$= (a \oplus m0) \oplus (b \oplus m1) \oplus (c \oplus m2) \oplus \ldots (z \oplus m25)$

$q1 = a1 \oplus b1 \oplus c1 \oplus \cdots \oplus z1 = m0 \oplus m1 \oplus m2 \oplus \cdots \oplus m25$

더 좋은 해결 : 마스크 두개를 교대하는 방식으로 안전한 마스킹 가능

$q0 = (a \oplus m0) \oplus (b \oplus m1) \oplus (c \oplus m1) \oplus \ldots (z \oplus m1) = a \oplus b \oplus c \oplus \cdots \oplus z \oplus m'$

$q1 = m0 \oplus m1 \oplus m1 \oplus \cdots \oplus m1 = m'$

masked AND

$q = a \wedge b = (a0 \oplus a1)(b0 \oplus b1) =$

$a0 \wedge b0 \oplus a0 \wedge b1 \oplus a1 \wedge b0 \oplus a1 \wedge b$

기존 마스킹 AND – 새로운 임의의 마스크 사용

$q0 = a0 \wedge b0 \oplus m2 \oplus a0 \wedge b1$

$q1 = a1 \wedge b0 \oplus m2 \oplus a1 \wedge b1$

Without Fresh Randomness

- Biryukov et al.의 새로운 임의의 마스크 추가가 필요 없는 기법
- q0 = a0 ∧ b0 ⊕ (a0 ∨ ¬b1)
  q1 = a1 ∧ b0 ⊕ (a1 ∨ ¬b1)
- q0 = a ⊕ b ⊕m1
  (a ⊕ b ⊕ m1) ⊕ m0 ⊕ m1 = a ⊕ b ⊕ m0 (마스크값으로 인해 안전)
- 위와 동일 경우
  Biryukov et al 의 방식은
  안전하지 않음

Shares			*Secrets*	*TT*		
a_0	b_0	b_1	b	q_0	$q_0 \oplus a_0$	$(q_0 \oplus a_0) \oplus b_0$
0	0	0	**0**	1	1	1
0	1	1		0	0	1
1	0	0		1	0	0
1	1	1		0	1	0
TT Hamming Weight:				**2**	**2**	**2**
0	0	1	**1**	0	0	0
0	1	0		1	1	0
1	0	1		1	0	0
1	1	0		0	1	1
TT Hamming Weight:				**2**	**2**	**0**

새로운 구조

- q0 =a0 ∧b0 ⊕(a0 ∨¬b1)
  =a0 ∧b0 ⊕¬(¬a0 ∧b1)
  =a0 ∧b0 ⊕(a0 ∧b1 ⊕b1)⊕1
- q1 =a1 ∧b0 ⊕(a1 ∨¬b1)
  =a1 ∧b0 ⊕¬(¬a1 ∧b1)
  =a1 ∧b0 ⊕(a1 ∧b1 ⊕b1)⊕1

Table 3: Truth table of the equation $(a_1 \wedge b_1 \oplus b_1)) \oplus a_1$

a_1	b_1	$= a_1 \vee b_1$
0	0	0
0	1	1
1	0	1
1	1	1

- q = a0 ∧ b0 ⊕ (a0 ∧ b1 ⊕ b1)) ⊕1 ⊕ a1
  ⊕ (a1 ∧ b0 ⊕ (a1 ∧ b1 ⊕ b1)) ⊕ 1 ⊕ a1

- q0 = (a0 ∧ b0 ⊕ (a0 ∧ b1 ⊕ b1)) ⊕ ((a1 ∧ b0 ⊕ (a1 ∧ b1 ⊕ b1)) ⊕ a1) = (a ∧ b) ⊕ m0
- q1 = a1 = m0

- (a1 ∧ b1 ⊕ b1)) ⊕ a1

새로운 구조

$$q_0 = \overbrace{(\underbrace{a_0 \wedge b_0}_{t_1} \oplus (\underbrace{a_0 \wedge b_1 \oplus b_1}_{t_2}))}^{t_3} \oplus \overbrace{(\underbrace{a_1 \wedge b_0}_{t_4} \oplus [m_0 \vee m_1])}^{t_5}$$

$$q_1 = a_1$$

Table 5: Security of the masked AND from Eqn. 7

Shares				*Secrets*			*TT*					
a_0	a_1	b_0	b_1	a	b	$a \wedge b$	t_1	t_2	t_3	t_4	t_5	q_0
0	0	0	0	**0**	**0**	**0**	0	0	0	0	0	0
0	0	1	1				0	1	1	0	1	0
1	1	0	0				0	0	0	0	1	1
1	1	1	1				1	0	1	1	0	1
TT Hamming Weight:							**1**	**1**	**2**	**1**	**2**	**2**
0	0	0	1	**0**	**1**	**0**	0	1	1	0	1	0
0	0	1	0				0	0	0	0	0	0
1	1	0	1				0	0	0	0	1	1
1	1	1	0				1	0	1	1	0	1
TT Hamming Weight:							**1**	**1**	**2**	**1**	**2**	**2**
0	1	0	0	**1**	**0**	**0**	0	0	0	0	1	1
0	1	1	1				0	1	1	1	0	1
1	0	0	0				0	0	0	0	0	0
1	0	1	1				1	0	1	0	1	0
TT Hamming Weight:							**1**	**1**	**2**	**1**	**2**	**2**
0	1	0	1	**1**	**1**	**1**	0	1	1	0	1	0
0	1	1	0				0	0	0	1	0	0
1	0	0	1				0	0	0	0	1	1
1	0	1	0				1	0	1	0	0	1
TT Hamming Weight:							**1**	**1**	**2**	**1**	**2**	**2**

적용

마스크 수동 추적은 복잡한 작업

- 자동화로 접근

- SMT 문제?

- Z3 사용, 각 게이트에 입력이 다른 마스크 같도록 하거나 원하는 마스크 같도록 설정

검증

AES- 128 함수의 입력과 출력 바이트의 마스크가 동일하게 적용

Cortex-M3과 Cortex-m4 사용 어셈블리 구현

2개의 부울 마스크 사용

CTR 모드에서 AES-128의 비트슬라이싱 병렬 구현

기존 4KB 암호화에 7,423 사이클, 이중 2,133 사이클(총 사이클의 29%)은
온보드 하드웨어 RNG 사용 10,496 비트 생성

제안 기법은 동일한 하드웨어 RNG 사용 32개의 비트 생성
28개를 무시하고 2개는 첫번째 AES, 2개는 두번째 AES

검증

게이트 수가 가장 적은 S-box 사용(113개)

AND 32개, XOR 77개, XNOR 4개

- 마스킹 후
  AND 96개, XOR 228개, XNOR 0개

- Yosys? 로 최적화
  AND 86개, XOR 225개, XNOR 4개

✓2.79배 오버헤드

검증

- ShitRow : 바이트 순서만 바뀌므로 고려 x
- Mixcolumns : 동일한 마스크가 결합되지 않도록해야하므로 기존 27개 XOR 마스크를 다시 마스크 필요
  - ➢ 12개의 XOR과 6개의 Load 명령어 추가
- AddRoundkey : 모든 바이트에 동리한 바이트 인코딩을 하므로 다시 마스킹 필요
  - ➢ 32개의 XOR 과 11개의 Load 명령어 그리고 1개의 Store 명령어

✓ 기존 보다 54% 속도향상 + 스텍 1,588바이트에서 188로 감소

Table 6: AES-128-CTR implementation results in terms of instruction counts

Module	*#*	*AND*	*XOR*	*Bit op**	*Load*	*Store*
AES†		870	3 433	560	773	520
PreRound	1	-	32	-	8	1
▶ AddRoundKey‡	1	-	32	-	8	1
Round	9	87	344	56	77	52
▶ SubBytes	1	87	225	-	60	51
▶ ShiftRows	1	-	48	56	-	-
▶ MixColumns	1	-	39	-	6	-
▶ AddRoundKey‡	1	-	32	-	11	1
LastRound	1	87	305	56	72	51
▶ SubBytes	1	87	225	-	60	51
▶ ShiftRows	1	-	48	56	-	-
▶ AddRoundKey‡	1	-	32	-	12	-

Table 7: Comparing Performance Results for AES-128

	Platform	*Speed [cycles]*	*Overhead factor*	*ROM [bytes]*	*RAM [bytes]*	*Random [bits]*
Comparable Platform						
This Work	Cortex-M4	3 387.6	2.1	25.2k	188	2
[SS16]	Cortex-M4	7 422.6	4.6	39.9k	2.0k	10.5k
[FPS17]	Cortex-M4	73 650	-	-	-	2/16
Different Platforms						
[RP10]	8-bit 8051	129 000	64.5	3.2k	73	9.6k
[BFG+17]	8-bit AVR	157 196	-	2.8k (in total)		13.1k
[BFG+17]	8-bit AVR	73 769	-	1.8k (in total)		11.5k
[GR17]	ARM7TDMI	53 462	-	7.5k	-	30.8k
[GR17]	ARM7TDMI	49 329	-	4.8k	-	26.9k
[GR17]	ARM7TDMI	56 199	-	12.4k	-	19.2k
[WVGX15]	Cortex-A15 simulator	4 869	4.3	-	-	19.2k

결론

- 기존 마스킹 대응기법의 대부분은 임의성의 양을 고려하지 않음
- 새로운 Masked AND의 제안과 검증을 통해 2비트의 무작위성만으로 마스킹 적용 가능함을 보임
- 2비트의 무작위 성을 요구한다면
  PRNG가 필요한지 온보드 TRNG를 사용할지 선택 가능
- 낮은 수준의 매우 제한된 장치에서 1차 마스킹 사용 가능

PIPO에 적용

- 최소화된 마스크 사용 하여 구현
  - PIPO는 8개의 sbox가 병렬로 연산되므로 2byte의 마스크 사용

- PIPO의 S-BOX 에는 6개의 AND연산과 5개의 OR 연산 사용되므로 Masked OR 필요
  - 효율성을 고려한 Masked OR 구조 제안

- 래퍼런스 코드 3,452사이클
  제안 기법 코드 3,612 사이클
  - 매우 적은 오버헤드

Masked OR

- Biryukov et al.의 Masked OR
  q0 = a0 ∧ b0 ⊕ (a0 ∨ b1)
  q1 = a1 ∨ b0 ⊕ (a1 ∧ b1)

q0 = (a0 ∧ b0 ⊕ (a0 ∨ b1)) ⊕ (a1 ∨ b0 ⊕ (a1 ∧ b1) ⊕ a1⊕ b1)
q1 = a1 ⊕ b1

q0 = a0 ∧ b0 ⊕ (a0 ∨ b1) ⊕ a1 ∨ b0 ⊕ (a1 ∨ b1)
q1 = a1 ⊕ b1

q0 = a0 ∧ b0 ⊕ (a0 ∨ b1) ⊕ a1 ∨ b0 ⊕ (a1 ∨ b1) ⊕ b1
q1 = a1

q0 = a0 ∧ b0 ⊕ (a0 ∨ b1) ⊕ a1 ∨ b0 ⊕ [(m0 ∨ m1)] ⊕ b1
q1 = a1 = m1

Masked OR

t5 t6

t1 t2 t3 t4

- q0 = (a0 ∧ b0 ⊕ (a0 ∨ b1)) ⊕ (a1 ∨ b0 ⊕ ([(m0 ∨ m1)] ⊕ b1))
- q1 = a1 = m1

Shares				*Secrets*	
a_0	a_1	b_0	b_1	a	b
0	0	0	0	**0**	**0**
0	0	1	1		
1	1	0	0		
1	1	1	1		
0	0	0	1	**0**	**1**
0	0	1	0		
1	1	0	1		
1	1	1	0		
0	1	0	0	**1**	**0**
0	1	1	1		
1	0	0	0		
1	0	1	1		
0	1	0	1	**1**	**1**
0	1	1	0		
1	0	0	1		
1	0	1	0		

t1	t2	t3	t4	t5	t6	q0
0	0	0	0	0	0	0
0	1	1	0	1	1	0
0	1	1	1	0	1	1
1	1	1	0	1	0	1
0	1	0	0	0	1	1
0	0	1	0	1	0	1
0	1	1	0	1	1	0
1	1	1	1	0	0	0
0	0	1	1	0	0	0
0	1	1	0	1	1	0
0	1	0	0	0	1	1
1	1	1	0	1	0	1
0	1	1	0	1	1	0
0	0	1	1	0	0	0
0	1	0	0	0	1	1
1	1	1	0	1	0	1

1 order Masking PIPO

- 라운드마다 표의 상태가 유지되도록 구현

	X[0]	X[1]	X[2]	X[3]	X[4]	X[5]	X[6]	X[7]
AddRK out	m0	m1	m0	m0	m1	m0	m1	m0
S-layer out	m2	m0	m1	m2	m1	m0	m0	m2
P-layer out	m2	ROL7(m0)	ROL4(m1)	ROL3(m2)	ROL6(m1)	ROL5(m0)	ROL1(m0)	ROL2(m2)

```
m[0] = m1;
m[1] = ((m0 << 7) | (m0 >> 1))^m1;
m[2] = ((m1 << 4) | (m1 >> 4))^m0;
m[3] = ((m2 << 3) | (m2 >> 5))^m0;
m[4] = ((m1 << 6) | (m1 >> 2))^m1;
m[5] = ((m0 << 5) | (m0 >> 3))^m1;
m[6] = ((m0 << 1) | (m0 >> 7))^m1;
m[7] = ((m2 << 2) | (m2 >> 6))^m0;
```

```
RK[0] ^= m2;
RK[1] ^= m2;
RK[2] ^= m2;
RK[3] ^= m2;
RK[4] ^= m2;
RK[5] ^= m2;
RK[6] ^= m2;
RK[7] ^= m2;

for (i = 8; i < (ROUND + 1) * 8; i++) {
   RK[i] ^= m[i % 8];
}
```

```
P[0] ^= m1;
P[1] ^= m0;
P[2] ^= m1;
P[3] ^= m1;
P[4] ^= m0;
P[5] ^= m0;
P[6] ^= m0;
P[7] ^= m1;
```

```
P[0] ^= m0;
P[1] ^= m1;
P[2] ^= m0;
P[3] ^= m0;
P[4] ^= m1;
P[5] ^= m1;
P[6] ^= m1;
P[7] ^= m0;
```

1 order Masking PIPO

- ADD RK : 마스크 초기화 단계에서 마스킹 된 상태이므로 추가 연산 X
- S-layer : AND 연산 17개, OR연산 5개, XOR 연산 44개 추가
- P-layer : 마스크 초기화 단계에서 마스킹 된 상태이므로 추가 연산 X

1 order Masking PIPO

- 레퍼런스 3,452 clock cycle
- 이전 결과 16,273 clock cycle
- 제안기법 3,612 clock cycle
  (함수호출 수 감소(약간 최적화 있음) -> 어셈블리로하면 더 감소)

Q & A