중복데이터를 이용한 16비트-MSP430 환경에서의 HIGHT 알고리즘 오류공격 대응 연구

고의석, 박보선, 서석충*

*국민대학교 정보보안암호수학과

HIGHT algorithm Against DFA with redundancy data on 16bit MSP430 enviornment

EuiSeok Ko, BoSun Park, Seog Chung Seo*

*Department of Information Security, Cryptology and Mathematics, Kookmin University

요 약

임베디드 기기의 무선 통신의 상용화가 진행되면서 임베디드 환경에서의 보안의 중요성이 강조되고 있다. 임베디드 기기는 암호 알고리즘이 동작 시 전력, 전자파, 시간 및 추가적인 정보를 누출하여 부채널 공격에 취약하다. 특히 물리적으로 오류를 주입하여 키를 복구하는 차분 오류 공격(Differential Fault Attack)에도 취약하다. 임베디드 기기에서 차분 오류 공격은 실용적인 부채널 공격이 되어 다양한 기기에서 키를 복구하는 연구들이 존재한다. 따라서 임베디드 기기에서 암호 알고리즘 구현 시 오류 공격 대응책은 필수적으로 적용해야 한다. 본 논문에서는 16비트 MSP 기기에서 HIGHT 암호의 오류 주입 공격에 대한 대응책을 제안한다. 제안하는 오류 공격 대응책은 하나의 레지스터에 내에 중복데이터를 적용하여 차분 오류 공격을 효과적으로 대응한다. *

I. 서론

무선 통신 기술의 발전으로 인해 각종 임베디드 기기는 이전보다 더 많은 데이터를 주고받아 사용자에게 편리함을 제공한다. 그러나 통신의 규모가 확대됨에 따라 해킹 및 정보의 노출, 변조 등 보안사고에 대한 예방의 필요성이 강조되고 있다. 통신 되는 데이터는 암호화가 적용되어야 한다. 임베디드 기기는 CPU 성능, 전력, 메모리 등 제한적인 환경이므로 기존의 블록암호가 아닌 제한된 환경에 적합한 경량암호를 사용해야 한다.

대표적인 경량암호 HIGHT는 2010년 ISO/IEC 국제표준에 지정된 제한된 환경에서 적합한 초경량/저전력 블록암호이다. 따라서 RFID/USN등과 같이 저전력 경량화를 요구하는 제한된 컴퓨팅 환경과 모바일 환경에 적합하다. HIGHT는 변형 Feistel 구조이며, 제한된 환경에 적합하도록 ARX(Addition-Rotation-XOR) 구조로 연산이 구성되어있다.

DFA(Differential Fault Attack, 차분 오류 공격)는 임베디드 기기에서 암호 알고리즘이 동작 시 오류를 주입하여 키를 복구하는 공격이다. 다양한 공격 시나리오가 존재하며, 공격자 가정이 높을수록 더 정밀한 위치와 시점에 오류를 주입할 수 있다. HIGHT, LEA, ARIA 등 다양한 블록암호에서 DFA 공격을 통해 키를 복구하는 연구들이 존재한다. 따라서 높은 수준의 보안을 제공하기 위해 구현 시 오류공격 대응책이 적용되어야 한다.

본 논문에서는 16비트 MSP430 환경에서 HIGHT 암호를 차분 오류 공격에 대하여 안전하게 암호화하는 구현 방안을 제안한다. 16비트 범용 레지스터에 중복 데이터를 적용하여 효과적으로 오류공격에 대응한다.

본 논문의 구성은 다음과 같다. 2장에서는

*이 논문은 2020년도 정부의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No.2019R1F1A1058494)

MSP430, 차분 오류 공격, HIGHT 암호를 설명한다. 3장에서는 제안하는 오류 공격 대응 구현 방법을 제시하고 대응 구현 성능을 분석한다. 4장에서는 결론을 제시하고 논문을 마무리한다.

II. 관련연구

2.1 MSP430

MSP430은 RISC(Reduced Instruction Set Computer) 기반의 48KB ROM, 10KB RAM을 가지는 16비트 프로세서이다.

MPS430은 General Purpose Registers를 12개를 갖는다. 8비트 AVR과 같은 다른 프로세서들에 비해 적은 개수의 General Purpose Registers를 갖기 때문에 레지스터에 대한 효율적 관리가 중요하다.

명령어	기능	설명	cc
ADD	src + dst → dst	Addition (8비트 or 16비트)	1
SUB	dst - src → dst	Subtract (8비트 or 16비트)	1
XOR	src ^ dst → dst	exclusive-or (8비트 or 16비트)	1
MOV	src → dst	Move data (8비트 or 16비트)	1
SWPB	dst 15:8 ↔ dst 7:0	swap byte (16비트)	1

[표 1] MSP430 Assembly Instruction set.
(cc : clock cycle, ^ : XOR 연산)

[표 1]에서는 HIGHT 암호화에 사용되는 MSP430의 명령어에 대한 설명과 clock cycle을 나타낸다. ADD, SUB, XOR, MOV는 8비트, 16비트에 대한 기능을 제공한다. 각 명령어의 기능은 [표 1]에 나와 있다. SWPB 명령어는 16비트 기능만 제공한다. SWPB 명령어는 x=(x<<4)^(x>>4)와 동일하다. 표에 나온 명령어의 clock cycle은 레지스터의 연산을 기준으로 작성하였다.

2.2 HIGHT 암호화

HIGHT는 64비트 길이의 평문을 128비트 키로 암호화하는 블록암호 알고리즘이다. 자원이 제한된 환경에서 경량화 구현이 기존 AES보다 성능이 뛰어나다.[1]

HIGHT의 암호화는 초기 변환, 라운드 함수, 최종 변환으로 구성된다. 초기 변환에서는 Whitening 키와 평문을 XOR 연산과 덧셈 연산을 통하여 라운드 함수의 입력값을 출력한다. 라운드 함수는 초기 변환에서 생성된 입력값과 서브키를 입력으로 받는다. 이 값들을 XOR 연산, 덧셈 연산, F함수 변환을 통한 라운드를 32번 진행한다. 덧셈 연산은 $GF(2^8)$에서의 연산을 의미한다. 최종 변환 과정은 이전 단계의 출력값과 whitening 키를 XOR와 덧셈 연산을 이용하여 암호문을 생성하는 과정이다.

2.3 차분 오류 공격

차분 오류 공격은 차분 공격과 부채널 공격 중 오류 주입 공격을 결합한 공격 방법이다. AES, ARIA, HIGHT 등 다양한 블록 암호 분석에 사용되고 있다.

본 논문에서 다루는 차분 오류 공격은 다음을 가정한다. 공격자는 특정 라운드의 입력레지스터에 임의의 1비트 오류를 주입할 수 있다. 단, 주입되는 위치는 선택할 수 없다. 오류를 주입 시킨 암호문과 주입 시키지 않은 정상적인 암호문을 모두 얻을 수 있다. 공격자는 이러한 가정 속에서 차분 확산 특성을 이용하여 오류가 주입된 비트의 위치를 알아낼 수 있다. 오류가 주입되었을 때, 가능한 차분 형태를 이용하여 불가능한 후보 키를 제거한다. 다수의 오류 암호문을 이용하여 후보 키를 충분히 줄인다면, 다항 시간 내에 키를 얻을 수 있다.

III. 차분 오류공격에 안전한 HIGHT 구현

3.1 레지스터 최적화

MSP430에서 레지스터 최적화의 필요성에 설명했다. 이 절에서는 레지스터 최적화 방법에 대한 방법을 서술한다.

암호화는 16바이트의 마스터키로부터 생성된

8비트 136개의 서브키의 주솟값, 8비트 평문 8개의 주솟값, F0 ,F1 주솟값을 입력으로 받는다. 사용 가능한 general registers가 12개(r4~r15)이기 때문에, 8비트 크기의 평문 8개를 레지스터에 저장하고, 4개의 레지스터를 평문, 라운드키, F0, F1의 인덱스 레지스터로 사용한다면, 사용 가능한 레지스터가 없다. 따라서 레지스터 최적화 과정이 필요하다. 기존 논문에서는 평문을 레지스터 8개에 모두 저장하고 push, pop을 이용하여 평문 인덱스 레지스터의 값을 비우는 방법을 사용한다. 본 논문에서는 거기에 추가로 F0와 F1 테이블을 합치는 방법을 통하여, 추가적인 레지스터를 확보한다. F0 테이블 256바이트와 F1 테이블 256바이트를 합쳐 512바이트인 새로운 테이블을 만든다. 만든다면, F0와 F1에 대한 인덱스 레지스터 두 개를 사용하지 않고, 하나의 인덱스 레지스터만을 사용하여도 된다. F1을 사용할 때 주솟값에 0x100(=256)을 더하고 F0를 사용할 때 빼는 방식으로 구현한다면 64 clock cycle 추가만으로 레지스터를 하나 더 확보할 수 있다.

레지스터	레지스터 활용
R4	첫번째 평문 블록
R5	두번째 평문 블록
R6	세번째 평문 블록
R7	네번째 평문 블록
R8	다섯번째 평문 블록
R9	여섯번째 평문 블록
R10	일곱번째 평문 블록
R11	여덟번째 평문 블록
R12	평문 인덱스 / 임시 레지스터
R13	서브키 인덱스
R14	F0, F1 테이블 인덱스
R15	임시 레지스터

[표 2] General Purpose registers의 사용 목적

3.2 MSP430 환경에서 오류 공격 대응 최적화 구현

중복을 사용하여 효과적으로 오류를 감지하는 구현을 제시한다. 중복을 적용하기 위해서는 하나의 레지스터에 같은 평문의 바이트가 중복으로 넣는다. 이 중복된 값을 비교하여, 오류가 주입되었을 시, 오류가 주입된 암호문을 출력하지 않게 한다.

MSP430 환경은 자체 데이터 병렬화를 지원하지 않는다. 따라서 중복을 사용한 차분 오류 공격에 안전한 구현은 더 많은 계산을 수반한다. 따라서 최적화 과정은 필수적이다. 기존 C 언어보다 속도가 빠른 어셈블리어로 구현하며 코드 최적화 역시 수행한다.

입력받은 하나의 8비트 평문을 두 개로 복사하여 16비트로 만들고 레지스터에 저장한다. 이 과정은 초기 변환과 동시에 진행할 수 있다. 초기 변환 중 XOR 연산은 비트 단위 연산이기 때문에 8비트 연산을 16비트로 바꿔주기만 하면 정상적으로 동작한다. 그러나 ADD 연산은 carry가 발생하기 때문에 별도의 처리 과정을 처리해야 한다. 하나의 8비트 키를 16비트의 중복된 평문 블록에 8비트 단위로 더하는 과정을 MSP430 환경에서 구현하면 다음 [그림 1]과 같다 .

```
INPUT : R12 = KEY , R5 = X1 || X2 (X, KEY : 8 bit)
OUTPUT : R5 = (X2+KEY) || (X1+KEY)
```

어셈블리 명령어	설명
1 : swpb r12	1 : r12 = KEY || 0x00
2 : add.w r12 r5	2 : r5 = r12+ r5 = (X1+KEY) || X2
3 : swpb r5	3: r5 = X2 || (X1 + KEY)
4 : add.w r12 r5	4 : r5 = r12 + r5 = (X2+KEY) || (X1+KEY)

[그림 1] ADD 병렬화 처리 과정

라운드 함수에 적용되는 ADD 연산과 XOR 연산은 앞선 방법들로 처리할 수 있다. 중복을 사용한 16비트 블록에 대한 F함수 과정을 구현하면 라운드 함수를 구현할 수 있다. F0 기준으로 다음 [그림 2]와 같다. F1의 경우, [그림 2]의 어셈블리 코드에서 F함수 TABLE 주솟값을 0x100과정만 추가 하면 된다.

```
INPUT : R14 = F table의 주소 , R5 = X1 || X2 (X : 8 bit), tmporal register(TMP1, TMP2)
OUTPUT : R12 = F[X1] || F[X2]
```

어셈블리 명령어	설명
1 : mov.b r5, r12	1 : r12 = X2
2 : add.w r12, r14	2~4 : tmp1 = F[x2]
3 : mov.b 0x0(r14), tmp1	5~6 : r12 = X1
4 : sub.w r12 r14	7~9 : tmp2 = F[X1]
5: swpb r5	10~12 : r12 = F[X1] || F[X2]
6 : mov.b r5, r12	
7 : add.w r12, r14	
8 : mov.b 0x0(r14), tmp2	
9 : sub.w r12 r14	
10. swpb tmp2	
11 : mov.b tmp1, r12	
12 : Xor.w tmp2, r12	

[그림 2] 16비트 블록에 대한 F함수 구현

F함수 구현 과정에 16비트 입력과 8비트 키의 덧셈 연산을 같이해주는 방법으로 중복되는 과정을 줄일 수 있다.

최종 변환까지 완료한 16비트 블록 전부에 대해 상위 8비트와 하위 8비트의 값이 같을 경우만 출력한다면, 차분 오류 주입 공격에 안전성을 확보할 수 있다. 8개의 블록을 각각 비교한다면, 많은 연산이 발생한다. 이 과정을 최적화하기 위하여 다음과 같이 수행한다. 모든 블록을 XOR 연산한다. 그 결괏값 상위 8비트와 하위 8비트도 XOR 연산하여 8비트로 만든다. 최종적으로 만들어진 8비트가 0일 경우, 암호문을 출력하고 0이 아닌 값을 가질 경우, 오류 주입 공격이 발생한 것으로 간주하여 출력하지 않는다.

4.3 성능 측정

기존 C언어로 구현된 한국인터넷진흥원의 오류 공격에 대응하지 않은 HIGHT 코드와 중복을 사용한 차분 오류 공격에 안전한 HIGHT 암호의 성능을 비교한다. IAR IDE의 simulator를 사용하여 clock cycle을 측정하였다. 측정 결과는 다음 [표 3]와 같다.

code	normal (KISA)	against DFA
CC	2629	3624 (+37.8%)

[표 3] clock cycle 측정
(CC : Clock Cycle)

clock cycle은 KISA C code와 비교하였을 때, 약 37.8% 증가하였다. 이는 37.8%의 속도 저하를 의미한다. HIGHT는 부채널 공격에 대응하는 기법을 적용할 경우, 많은 성능 저하가 나타난다. 본 논문에서는 이를 최소화하였다.

IV.결론

본 논문에서는 차분 오류 공격에 대응하는 방법을 제시하고, MSP430 환경에서 구현하였다. MSP430은 적은 General Purpose registers를 갖기 때문에 레지스터 최적화를 진행하였다. 공격에 대응하기 위해 발생하는 많은 연산 부하를 최소화하기 위해 구현 최적화를 하였다. 결과적으로 차분 오류 공격에 대응하는 HIGHT 암호를 성능 저하를 최소화하여 구현하였다.

이후 연구 목표로는 다양한 오류 공격에 대응하는 방안과 다양한 암호 알고리즘에 적용하는 방법을 연구할 계획이다.

[참고문헌]

[1] Hong, D., et al.: HIGHT: a new block cipher suitable for low-resource device. In: Goubin, L., Matsui, M. (eds.) CHES 2006 .LNCS, vol. 4249, pp. 46 - 59. Springer, Heidelberg.

[2] H.J.Seo, K.H.An, H.D.Kwon, Compact LEA and HIGHT Implementations on 8-비트 AVR and 16-비트 MSP Processors, International Workshop on Information Security Applications, 253-265

[3] 한국인터넷진흥원, KISA(Korea Internet & Security Agency) 레퍼런스 소스코드 : https://seed.kisa.or.kr/kisa/Board/18/detailView.do

[4] 한국인터넷진흥원, HIGHT 블록암호 알고리즘 사양 및 세부 명세서 : http://seed.kisa.or.kr/kor/hight/hightInfo.jsp

[5] TEXAS INSTRUMENTS, MSP430x5xx and MSP430x6xx Family User's Guide : https://www.ti.com/lit/ug/slau208q/slau208q.pdf?ts=1603290751075&ref_url=https%253A%252F%252Fwww.google.com%252F

[6] 송진교, 서석충, ARMv8에서 ASIMD 명령어를 이용한 HIGHT 암호 최적화 연구, 2020정보보호학회 하계학술대회

[7] 이유섭, 김종성, 홍석희, 블록 암호 HIGHT에 대한 차분 오류 공격, 정보보호학회논문지 22(3), 485-494