부채널 분석 대응을 위한 1 차 마스킹 AES 알고리즘 최적화 구현

김경호*, 서화정*
*한성대학교 IT 융합공학과
e-mail : pgm.kkh@gmail.com

Implementation of Optimized 1st-Order Masking AES Algorithm Against Side-Channel-analysis

Kyung-Ho Kim*, Hwa-Jeong Seo*
*Dept. of Applied IT Engineering, Han-Sung University

요 약

최근 사물인터넷 기술의 발전과 함께 하드웨어 디바이스에서 측정하는 센싱 데이터를 보호하기 위해 다양한 방식의 암호화 알고리즘을 채택하고 있다. 그 중 전 세계에서 가장 많이 사용하는 암호화 알고리즘인 AES(Advanced Encryption Standard) 또한 강력한 안전성을 바탕으로 많은 디바이스에서 사용되고 있다. 하지만 AES 알고리즘은 DPA(Differential Power Analysis), CPA(Correlation Power Analysis) 같은 부채널 분석 공격에 취약하다는 점이 발견되었다. 본 논문에서는 부채널 분석 공격 대응방법 중 가장 널리 알려진 마스킹 기법을 적용한 AES 알고리즘의 소프트웨어 최적화 구현 기법을 제시한다.

1. 서론

사물인터넷 기술의 발전으로 많은 디바이스들이 하나의 네트워크 안에서 연결되어 수많은 정보를 서로 주고받을 수 있게 되었다. 이러한 네트워크에서는 데이터를 안전하면서도 빠르게 송수신할 수 있어야 한다. 이를 위해서 현재 대부분의 사물인터넷 기기에 AES 암호화 알고리즘을 포함한 다양한 종류의 경량 암호화 알고리즘을 적용하여 데이터를 빠르고 안전하게 전송하고 있다. [1,2]

그러나 암호 알고리즘 자체의 취약점이 아닌 하드웨어의 전력 소모량 분석, 시차 분석[3], 오류 주입[4], 전자기파 분석[5] 등 다양한 방식의 부채널 정보를 이용하여 암호화된 정보를 탈취할 수 있다. 이러한 부채널 분석에 대응하기 위해 마스킹[6,7], 랜덤 지연시간 삽입[8] 등의 다양한 대응 방안을 연구하고 있고 그 중 마스킹을 이용한 대응 기법이 가장 많이 연구되는 분야이다.

마스킹 대응 기법은 암호화 알고리즘의 특정 연산 마다 암호화할 데이터와 난수 값을 XOR(eXclusive OR) 연산을 이용하여 데이터를 변환해서 공격자가 전력 분석을 할 수 없도록 하는 방법이다. 더 많은 연산을 해야 하기 때문에 마스킹 된 AES 알고리즘의 암호화 속도는 마스킹 하지 않은 AES 알고리즘에 비해 느리지만 전력분석을 이용한 부채널 분석 공격에 대한 안전성을 확보할 수 있다. 본 논문에서는 부채널 분석 공격에 대응하기 위한 1 차 마스킹 기법을 AES 알고리즘에 적용하고 다른 경량 암호에 비해 암호화 시간이 긴 AES 알고리즘의 단점을 보완하기 위한 소프트웨어 최적화 구현 기법을 제시한다.

본 논문의 구성은 다음과 같다. 2 장에서는 전력 분석을 이용한 부채널 공격 방법 소개와 대응 방안인 1 차 마스킹 기법을 적용한 AES 알고리즘에 대해 설명하고 3 장에서는 제안하는 최적화 구현 기법을 제시한다. 4 장에서는 최적화 된 AES 알고리즘의 성능 분석 결과를 확인해본다. 마지막으로 5 장에서는 본 논문의 결론을 내린다.

2. 전력 분석을 이용한 공격과 1 차 마스킹 기법

본 장에서는 전력 소모량 분석을 이용한 부채널 공격으로 가장 잘 알려진 DPA[9]와 CPA[10] 공격에 대한 설명과 그에 대한 대응 방안인 1 차 마스킹 기법을 적용한 AES 암호화 알고리즘에 대해서 서술한다.

2.1 전력 분석을 이용한 부채널 공격

부채널 분석은 암호화 알고리즘을 사용하는 디바이스의 전력 소모량의 변화를 관찰하여 통계적인 분석을 통해 키 값을 알아낸다. 즉 디바이스가 키 값과 관련이 있는 데이터의 0 과 1 값을 연산할 때 두 값의 전력 사용량의 차이를 이용한다.

가장 많이 사용되는 전력 분석을 이용한 부채널 공격 방식으로는 DPA 와 CPA 공격 기법이 있다. DPA 와 CPA 는 암호화 알고리즘이 적용된 디바이스에 동일한 키 값을 이용해서 다른 평문을 연속적으로 입력하여 암호문을 얻고 전력 분석 모듈을 이용하여 파형을 수집한다. 수집된 정보들을 바탕으로 정해진 분류 함수를 이용하여 파형 통계 처리로 키 값을 분석하는 방식이다.

DPA 는 평문과 분석하는 부분의 출력 값 등의 정보를 이용하여 결과 값을 0 과 1 그룹으로 파형을 나누고 그룹 간 평균값을 구하여 두 그룹의 평균값의 차이로 키 값을 분석하는 방식이고 CPA 는 알고리즘의 특정 연산 부분을 지정하여 분석한 다음 분석된 데이터를 이용하여 중간값 추정치를 예측한다. 추정치를 이용하여 예상되는 전력 소모 모

델을 설계하고 실제 전력소모 모델과 비교하여 상관 계수가 높은 값을 키 값으로 분석한다.

2.2 1 차 마스킹 AES

기본적인 AES-128 알고리즘은 128bit 의 키 값을 이용하여 10 번의 라운드에 걸쳐서 사용할 라운드 키를 계산하고 AddRoundKey, SubBytes, ShiftRows, MixColumns 연산을 반복하여 암호문을 얻게 되는 전 세계적으로 가장 많이 사용하는 대칭키 암호화 알고리즘이다.

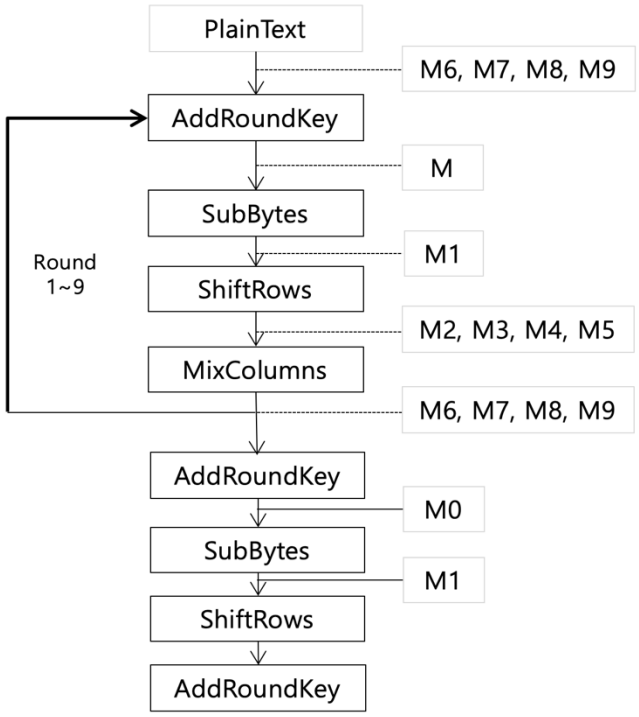
이러한 AES 알고리즘의 암호화 과정에서 공격자는 전력분석을 통해 CPA 공격을 이용하여 특정 연산의 데이터 값을 분석하고 전력 소모 모델을 만들어서 상관 계수가 높은 키 값을 찾아낸다. 따라서 CPA 공격에 대응하기 위하여 임의 값으로 이루어진 마스킹 값을 암호화 과정마다 반복적으로 적용해 전력분석을 방해하여 공격자의 CPA 공격을 방어할 수 있다.[11]

마스킹 AES 알고리즘을 보기 쉽게 도식화하면 (그림 1)과 같다. 우선 마스킹 적용을 위해서 AES 알고리즘이 시작될 때 6 개의 난수 값을 이용하여 M0, M1, M2, M3, M4, M5 변수를 만든다. 그리고 M2, M3, M4, M5 값을 이용하여 MixColumns 연산을 통해 나온 값을 M6, M7, M8, M9 값으로 사용한다. 그 다음 SubBytes 연산에서 사용할 Sbox 테이블을 MaskedSBOX[i ^ M0] = SBOX[i] ^ M1 값으로 마스킹 한다. 이제 생성된 10 개의 임의의 마스크 값(M0 ~ M9)과 마스킹 된 Sbox 테이블을 이용해 라운드 별로 이루어지는 특정 연산에 마스킹 연산을 추가한다.

첫 번째 라운드를 시작하기 전에 (4 x 4) 행렬의 암호화할 평문(암호문)을 첫번째 행은 M6, 두번째 행은 M7, 세번째 행은 M8, 마지막 행은 M9 으로 XOR 연산을 수행한다. 그리고 첫 번째 라운드부터 AddRoundKey 연산에서 암호문에 마스킹된 M6, M7, M8, M9 값을 동일한 값으로 XOR 연산을 수행하여 M6, M7, M8, M9 값을 제거하고 다시 모든 암호문을 M0 값으로 마스킹 한다. M0 값으로 마스킹 된 암호문은 SubBytes 연산에서 사전에 만든 마스킹된 Sbox 테이블을 참조하면서 M0 값이 제거되고 M1 값으로 마스킹 된다. 그 이후 ShiftRows 연산에선 특별한 마스킹 연산이 이루어지지 않다가 MixColumns 연산이 이루어지기 전에 모든 암호문에 마스킹 되어있는 M1 값을 제거하고 첫 번째 행은 M2, 두 번째 행은 M3, 세 번째 행은 M4, 마지막 행은 M5 값으로 다시 마스킹 한다. 마지막으로 MixColumns 연산 과정에서 M2, M3, M4, M5 값이 자연스럽게 M6, M7, M8, M9 으로 변경된다. 이와 같이 AddRoundkey – SubBytes – ShiftRows – MixColumns 의 연산 과정을 9 번의 라운드 동안 반복하기 때문에 9 번의 라운드가 끝난 뒤에는 M6, M7, M8, M9 의 값이 마스킹된 암호문이 생성된다.

마지막 10 라운드에서 AddRoundkey – SubBytes – ShiftRows 연산을 통해 암호화 과정을 마무리하면 암호문에 M1 값이 마스킹 값으로 적용된다. 마지막으로 이전에 사용하던 AddRoundKey 연산에서 M6, M7, M8, M9 마스킹 연산을 포함하지 않는 AddRoundKey 연산을 이용하여 M1 값을 제거하면서 모든 마스킹 연산이 무효화된 암호문을 얻을 수 있다.

결과적으로 마스킹 값은 각 라운드마다 암호화 연산 과정에서 같은 값과 XOR 연산이 이루어지기 때문에 자연스럽게 사라지며 AES 알고리즘의 결과 값인 암호문은 마스킹을 적용하지 않은 암호문과 동일하지만 암호화 과정에서 필요없는 연산들을 추가하여 전력 소모량 측정을 방해하는 효과를 얻을 수 있다.



(그림 1) Masking AES

3. 최적화 구현 기법

본 장에서는 기존의 AES 알고리즘의 연산 방식을 효율적으로 바꾸고 Inline Assembly 를 활용하여 소프트웨어 최적화 구현을 제시한다.

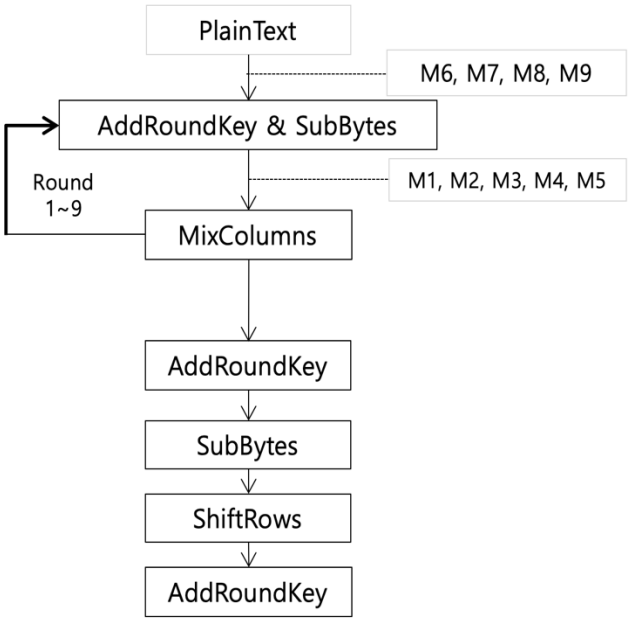
3.1. 사전 Key 연산

AES 알고리즘은 라운드 별로 사용하는 키 값이 다르기 때문에 라운드 마다 이전 라운드에서 사용한 키 값을 이용하여 새로운 라운드 키 값을 생성한다. 이 과정은 반복되는 라운드 마다 계산을 해야 하는 오버헤드를 가지는데 해당 연산을 AES 알고리즘이 최초 시작할 때 한번의 연산 과정으로 실제 키 값을 포함하여 총 11 세트의 키를 생성한다. 또한 키 값이 사용되는 연산은 AddRoundKey 연산 밖에 없기 때문에 해당 연산에서 사용되는 마스킹 값인 M6, M7, M8, M9 값과 M0 값을 미리 라운드 키 값에 계산하여 라운드 마다 마스킹 연산을 추가해야 하는 오버헤드를 줄일 수 있는 장점이 있다.

3.2. 연산 과정 통합

최적화된 AES 를 도식화하면 (그림 2)와 같다. AddRoundkey – SubBytes – ShiftRows - MixColumns 연산을 반복적으로 수행하는 AES 알고리즘을 보다 효율적으로 계산하기 위해서 AddRoundKey 연산과 SubBytes 연산을 하나의 반복문 함수에 포함시켜 구현하였고 SubBytes 연산이 끝난 뒤 M2, M3, M4, M5 값을 마스킹 연산해야 하는데 이 과정 또한 같은 반복문에서 적용함으로써 메모리에 두 번 접근해야 하는 오버헤드를 줄일 수 있다.

또한 ShiftRows 연산의 경우 배열에 저장된 값의 순서를 바꿔야 하는 비교적 오버헤드가 큰 연산이기 때문에 해당 연산을 하지 않고 MixColumns 연산에서 배열의 인덱스를 이용하여 ShiftRows 연산이 된 것처럼 연산을 수행하여 명령어 수와 연산 시간을 줄일 수 있는 장점이 있다.



(그림 2) 최적화된 AES

3.3. Inline Assembly 구현

C 언어와 같은 고급 언어로 암호화 알고리즘을 구현할 경우 컴파일 과정에서 불필요한 연산이 의도치 않게 들어가는 경우가 있다. 이러한 불필요한 연산은 연산 시간을 늘리고 명령어 수 또한 늘리기 때문에 성능에 민감한 암호화 알고리즘은 Assembly 언어로 작성하는 것이 최적화 구현에 적합하다.

Assembly 언어 같은 경우 사용하는 하드웨어에 종속되기 때문에 하드웨어의 특성을 잘 파악해야 한다. 본 논문에서 사용하는 Atmel 사의 Atmega128 보드 같은 경우 32 개의 8bit 레지스터를 사용하여 연산 및 데이터 Load 및 Store 할 수 있고 16bit 주소 체계를 사용하기 때문에 정해진 8bit 레지스터 2 개를 같이 사용하여 16bit 포인터 연산을 할 수 있다.

AES 알고리즘의 경우 대부분의 연산 시간을 라운드 함수에서 사용함으로 해당 부분을 Inline Assembly 를 이용하여 레지스터에 비해 액세스 속도가 느린 메모리 접근을 최소화하고 대부분의 연산을 레지스터를 이용함으로써 속도를 향상시킬 수 있다. 또한 컴파일 과정에서 생기는 불필요한 명령어 코드들을 제거하여 명령어 코드 수를 줄일 수 있는 장점이 있다.

4. 성능 분석

본 논문에서 제시하는 최적화 방식의 Clock Cycle 및 명령어 수는 Atmel Studio 의 Atmega128 시뮬레이터를 이용하여 측정하였으며 직접 C 언어로 구현한 일반적인 AES 알고리즘 및 1 차 마스킹이 적용된 AES 알고리즘 성능과 비교한다.

마스킹 연산을 포함하지 않는 AES 알고리즘, 마스킹 연산을 포함한 AES 알고리즘 그리고 마스킹 연산을 포함한 최적화된 AES 알고리즘의 명령어 수 및 실행 시간을 비교하고 1 차 CPA 공격을 시도하여 최적화된 1 차 마스킹 AES 알고리즘이 CPA 공격으로부터 데이터를 보호하고 암호화 속도 또한 C 언어로 구현된 AES 알고리즘 보다 효율적임을 증명한다.

<표 1>은 C 언어로 구현된 일반적인 AES 알고리즘과 Masked AES 그리고 Inline Assembly 로 구현된 Masked AES 알고리즘의 Clock Cycle 을 나타낸다. C 언어로 구현된 Masked AES 알고리즘에 비해 Assembly 로 구현된 AES 알고리즘이 훨씬 적은 Clock Cycle 을 보이는 것을 볼 수 있다.

<표 1> 암호화 방식에 따른 Clock Cycle 비교

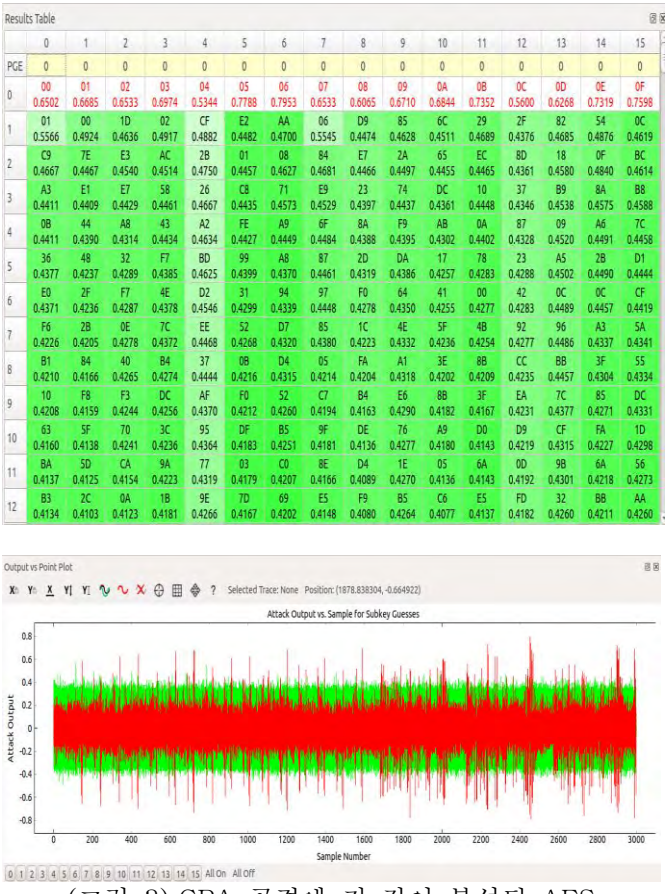
	AES	Masked AES	Assembly Masked AES
Clock Cycle	22706	29269	25970

<표 2>는 암호화 방식에 따른 라운드 함수 내부의 명령어의 수를 보여준다. 마스킹 연산이 들어가는 Masked AES 는 일반적인 AES 에 비해 명령어 수가 많아지는 것을 확인할 수 있고 Assembly Masked AES 는 컴파일 과정에서 불필요한 연산들을 지우고 최적화 연산을 이용하여 명령어 수가 확연히 줄어든 것을 확인할 수 있다.

<표 2> 암호화 방식에 라운드 함수 명령어 수 비교

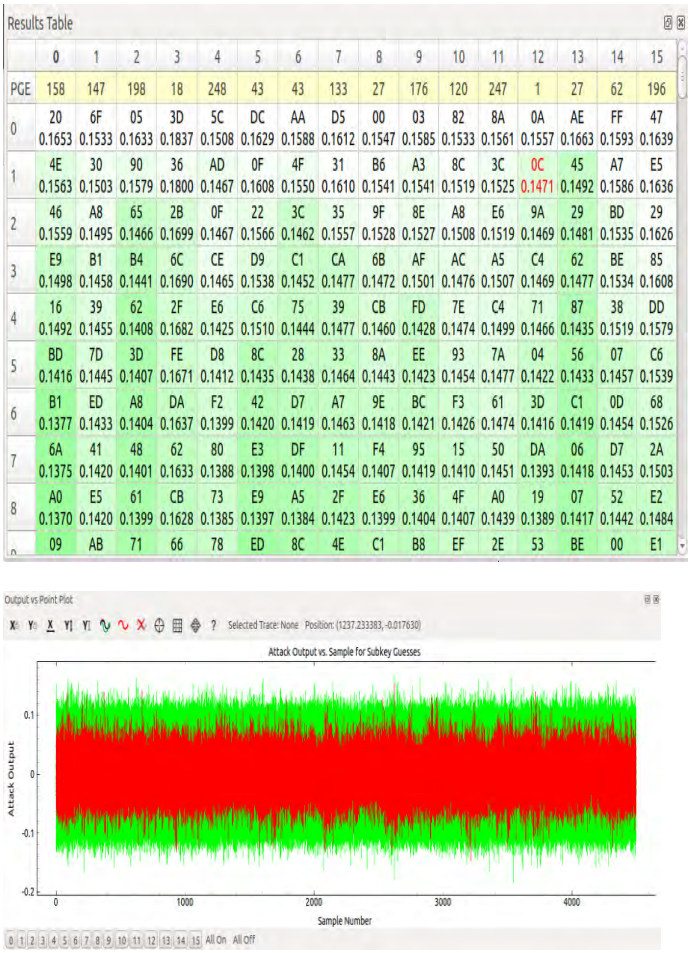
	AES	Masked AES	Assembly Masked AES
명령어 수	290	400	191

(그림 3)에서는 마스킹이 적용되지 않은 AES 알고리즘에 CPA 공격을 시도하여 상관 계수가 상당히 높게 정확한 키 값이 분석된 것을 확인할 수 있다. 아래의 파형을 살펴보면 특정 주기마다 파형이 크게 튀는 것을 볼 수 있는데 해당 지점에서 키 값이 분석된 것이다.



(그림 3) CPA 공격에 키 값이 분석된 AES

(그림 4) 에서는 마스킹 연산이 이루어진 AES 알고리즘에 CPA 공격을 해도 큰 상관 계수를 보인 (그림 3)에 비해서 상관 계수에 큰 차이가 없어서 특정 키 값을 찾아낼 수 없는 것을 확인할 수 있다. 13 번째 키의 경우 실제 키 값의 상관도가 2 번째로 높게 분석되었지만 다른 예상 키 값의 상관도와 큰 차이가 없기 때문에 공격자가 실제 키 값으로 판단하기엔 어려움이 있다. 뿐만 아니라 아래 파형을 살펴보아도 (그림 3)의 파형에 비해 크게 튀는 부분 없이 전체적으로 비슷한 모양의 파형이 반복되는 것을 확인할 수 있다.



(그림 4) CPA 공격을 방어한 Masking AES

5. 결론

본 논문에서는 AVR 프로세스 상에서 부채널 분석 공격에 대응하기위해 AES 암호화 알고리즘에 1 차 마스킹 기법을 적용하였고 Inline Assembly 를 이용한 소프트웨어 최적화 구현 기법에 대해 알아보았다. 이는 부채널 분석 공격에 취약점을 보인 기존의 AES 알고리즘을 보완하였고 Inline Assembly 를 이용하여 명령어 수 및 실행 시간에서 C 언어에 비해 좋은 성능을 보임을 증명하였다.

AES 알고리즘은 다른 경량 암호화 알고리즘에 비해 연산량이 많아서 비교적 많은 실행 시간이 필요하기 때문에 사물인터넷 디바이스에서 많이 사용되고 있지는 않다. 하지만 아직까지도 많은 연구가 나오고 있는 만큼 전력분석을 이용한 부채널 공격 대응 방법 및 Inline Assembly 를 이용한 최적화 소프트웨어 구현은 의미가 있다.

본 논문에서 제안하는 소프트웨어 최적화 구현을 토대로 앞으로의 연구 방향은 Assembly 를 이용한 효율적인 암호화 구현을 제시하는 데 있다.

참고문헌

[1] 문시훈, 김민우, 권태경. "IoT 통신 환경을 위한 경량 암호 기술 동향." The Journal of The Korean Institute of Communication Sciences, 33.3 (2016.2): 80-86.

[2] Hwajeong Seo, Howon Kim. "사물인터넷을 위한 경량 암호 알고리즘 구현." REVIEW OF KIISC, 25.2 (2015.4): 12-19.

[3] P.C. Kocher, “Timing Attacks on Implementation of Diffe-Hellman, RSA, DSS and Other Systems,” Proc. Adv. Cryptology, 1996, pp. 104-113.

[4] E. Biham and A. Shamir, “Differential Fault Analysis of Secret Key Cryptosystems,” Proceedings of Crypto 1997, LNCS 1294, pp. 513-525, Aug. 1997.

[5] J. Quisquater and D. Samyde, “Electromagnetic Analysis (EMA): Measures and Countermeasures for Smart Cards,” Proc. e-Smart, 2001, pp. 200-210.

[6] HyungSo Yoo, JaeCheol-Ha, ChangKyun Kim, IlHwan Park, SangJae Moon. "A Secure ARIA implementation resistant to Differential Power Attack using Random Masking Method." Journal of the Korea Institute of Information Security & Cryptology, 16.2 (2006.4): 129-139.

[7] ChangKyun Kim, JaeHoon Park, Daewan Han, Dong Hoon Lee. "Investigation of Masking Based Side Channel Countermeasures for LEA." Journal of the Korea Institute of Information Security & Cryptology, 26.6 (2016.12): 1431-1441.

[8] L.Goubin and J.Patarin, " DES and Differential Power Analysis - The Duplication Method," CHES 1999, LNCS 1717, pp.158-172, Springer, 1999

[9] Paul Kocher, Joshua Jaffe, and Benjamin Jun, "Differential Power Analysis," CRYPTO '99, Springer-Verlag, 1999, pp.388-397

[10] Eric Brier, Christophe Clavier, Francis Olivier, “Correlation Power Analysis with a Leakage Model,” CHES 2004: Cryptographic Hardware and Embedded Systems - CHES 2004, pp 16-29

[11] Herbst, C., Oswald, E., & Mangard, S. (2006). An AES Smart Card Implementation Resistant to Power Analysis Attacks. Lecture Notes in Computer Science, 239–252.