석사학위논문

저전력 프로세서 상에서의 AES-GCM 최적화 구현

2021년

한 성 대 학 교 대 학 원

I T 융 합 공 학 과

I T 융 합 공 학 전 공

김 경 호

석 사 학 위 논 문
지도교수 서화정

저전력 프로세서 상에서의 AES-GCM 최적화 구현

Optimized implementation of AES-GCM
on Low-End Microcontrollers

2020년 12월 일

한 성 대 학 교 대 학 원

I T 융 합 공 학 과

I T 융 합 공 학 전 공

김 경 호

석 사 학 위 논 문
지도교수 서화정

저전력 프로세서 상에서의 AES-GCM 최적화 구현

Optimized implementation of AES-GCM
on Low-End Microcontrollers

위 논문을 공학 석사학위 논문으로 제출함

2020년 12월 일

한 성 대 학 교 대 학 원

I T 융 합 공 학 과

I T 융 합 공 학 전 공

김 경 호

김경호의 공학 석사학위 논문을 인준함

2020년 12월 일

심사위원장 _____(인)

심 사 위 원 _____(인)

심 사 위 원 _____(인)

국 문 초 록

저전력 프로세서 상에서의 AES-GCM 최적화 구현

한 성 대 학 교 대 학 원
I T 융 합 공 학 과
I T 융 합 공 학 전 공
김 경 호

본 논문은 전 세계적으로 가장 많이 사용되는 블록 암호 알고리즘 AES(Advanced Encryption Standard)의 대표적인 운영모드인 Galois Counter Mode(GCM)를 AVR 8-bit 저전력 프로세서 상에서 최적화하여 구현할 수 있는 기법에 대해 서술한다. AES-GCM은 데이터의 기밀성을 유지하고 송신자를 인증할 수 있는 인증성을 갖춰 데이터를 송수신할 때 가장 많이 사용되는 암호 운영모드이다. 본 논문에서는 AES-GCM을 최적화 구현하기 위해 세 가지 최적화 기법을 제안한다. 첫째, AES-GCM에서 암호화를 담당하는 AES-CTR 모드의 복잡한 암호화 과정을 사전 연산을 통한 테이블 참조 연산으로 대체하여 기존 방식에 비해 약 20% 성능을 향상한다. 둘째, AES-GCM에서 인증성을 담당하는 128-bit 다항식 곱셈 연산으로 이루어진 GHASH 연산에 대해 레지스터 사용 최소화 및 Karatsuba 곱셈 알고리즘을 이용한 최적화 구현기법을 제안한다. 마지막으로 암호화 과정에서 사용되는 물리적인 전력 사용량을 분석하여 암호 알고리즘을 공격하는 SPA(Simple Power Analysis), DPA(Differential

Power Analysis), CPA(Correlation Power Analysis) 등의 부채널 공격에 대한 대응성을 갖추기 위해 마스킹 연산을 최적화하여 구현함으로써 부채널 공격에 대한 대응성을 증명한다.

【주요어】 AES-GCM, 최적화 구현, 부채널 공격, Karatsuba 알고리즘, 저전력 프로세서

목 차

표 목 차

알 고 리 즘 목 차

그 림 목 차

Ⅰ. 서 론

1.1 연구 목적

4차 산업혁명 시대에 들어서면서 IOT(Internet of Things), 인공지능 등의 기술의 발전으로 정보의 중요성이 부각되고 있다. 따라서 정보를 지키기 위한 다양한 암호 알고리즘 개발 및 암호와 관련된 기술에 대한 연구가 진행되고 있다. 특히 저전력 프로세서에서 다양한 종류의 민감한 정보들을 서버에 전송해야하는 IOT 환경의 경우 이러한 정보들을 빠르고 안전하게 보내는 것이 중요하다. 따라서 최근에는 암호화 연산시간이 비교적 긴 기존 블록 암호들을 대신해 IOT 환경에서 사용할 수 있는 다양한 종류의 경량 암호 및 정보의 기밀성과 송신자를 인증할 수 있는 인증성을 함께 갖춘 AEAD(Authenticated Encryption with Associated Data) 종류의 암호가 개발되고 있다. 하지만 이러한 암호들은 AES와 같은 기존 블록 암호에 비해 암호화 연산이 간단하기 때문에 보안성이 떨어진다.

2018년 CHES에서는 강력한 보안성을 갖지만 암호화 연산 속도가 느린 AES-CTR 모드를 사전 연산 테이블을 이용하여 최적화 하는 FACE(Fast AES-CTR Encryption) 기법에 대해서 제안했다. 하지만 5KB 이상의 메모리가 필요한 FACE 기법의 경우 IOT 환경에서 흔히 사용되는 8-bit 저전력 프로세서에서는 적용할 수 없다.

본 논문에서는 AES 암호 알고리즘에 AEAD를 적용시킨 AES-GCM에 대한 최적화 구현기법을 제안한다. 우선 AES-GCM에서 암호화 부분을 담당하는 AES-CTR 모드를 FACE 기법의 단점을 보완하여 8-bit 저전력 프로세서에도 사용할 수 있도록 경량화하는 최적화 기법을 제안한다. 그리고 AES-GCM에서 인증성을 담당하는 GHASH 연산에 포함된 128-bit 이진 곱셈 연산에 대한 최적화 구현 기법을 제안한다. 마지막으로 대표적인 부채널 공격인 전력 파형 분석에 대한 저항성을 가지기 위해 마스킹 연산을 적용

하여 전력 파형 공격에 대한 저항성을 증명한다.

1.2 연구 기여

본 논문의 기여는 다음과 같다.

첫째, AES-CTR 암호화 모드에 대해 Intel 프로세서와 같은 고사양 프로세서에 한정된 이전 연구들을 저전력 프로세서인 8-bit AVR 프로세서에서 사용할 수 있도록 경량화하여 제안한다. 이전 연구인 FACE 기법의 경우 저전력 프로세서의 메모리 크기에서는 사용할 수 없는 5KB의 사전 연산 테이블을 위한 메모리 크기가 필요하고 256개 블록의 암호화 연산 이후에는 사전 연산 테이블을 업데이트해야하는 과정이 필요하다. 반면에 제안하는 기법은 사전 연산 테이블 업데이트 과정이 필요 없을 뿐만 아니라 저전력 프로세서에 맞게 명령어와 레지스터 사용을 최적화하여 8-bit AVR 프로세서에서 128/192/256-bit 기준 바이트 당 138/168/199 cc(Clock Cycle)의 암호화 연산 시간을 보인다. 또한 CPA, DPA와 같은 전력 소비 파형을 이용한 부채널 공격에 대응성을 갖추기 위해 1차 마스킹 연산을 추가한 최적화 구현 기법을 제안한다.

둘째, AES-CTR 모드의 2 라운드까지 사전 연산을 하는 이전 연구에서 3라운드 SubBytes 연산과 ShiftRows 연산을 추가로 확장하여 기존 연구에 비해 암호화 연산 시간을 단축하는 최적화 구현 기법을 제안한다. 해당 기법을 이용하면 8-bit AVR 프로세서에서 128/192/256-bit 기준 바이트 당 122/ 153/183 cc의 암호화 연산 시간으로 성능을 향상시킬 수 있다.

셋째, AES-GCM의 GHASH 연산에서 사용되는 128-bit 이진 곱셈 연산을 Karatsuba Block-Comb 방식을 적용하여 연산 시간을 최적화한다. 또한 위장 레지스터 연산을 추가하여 TA(Timing Attack), SPA(Simple Power Analysis)와 같은 부채널 공격에 대응성을 갖춘다.

1.3 표기법

본 논문에서 사용하는 표기법은 [표 1-1]과 같다. i는 바이트의 순서, j는 라운드, S[i]는 각 라운드의 입력 값 또는 출력 값을 의미한다. 마지막으로 M[i]는 각 바이트마다 전력 파형 분석 공격에 대응하기 위해 사용되는 무작위 난수 값인 마스킹 값을 의미한다.

[표 1-1] 표기법

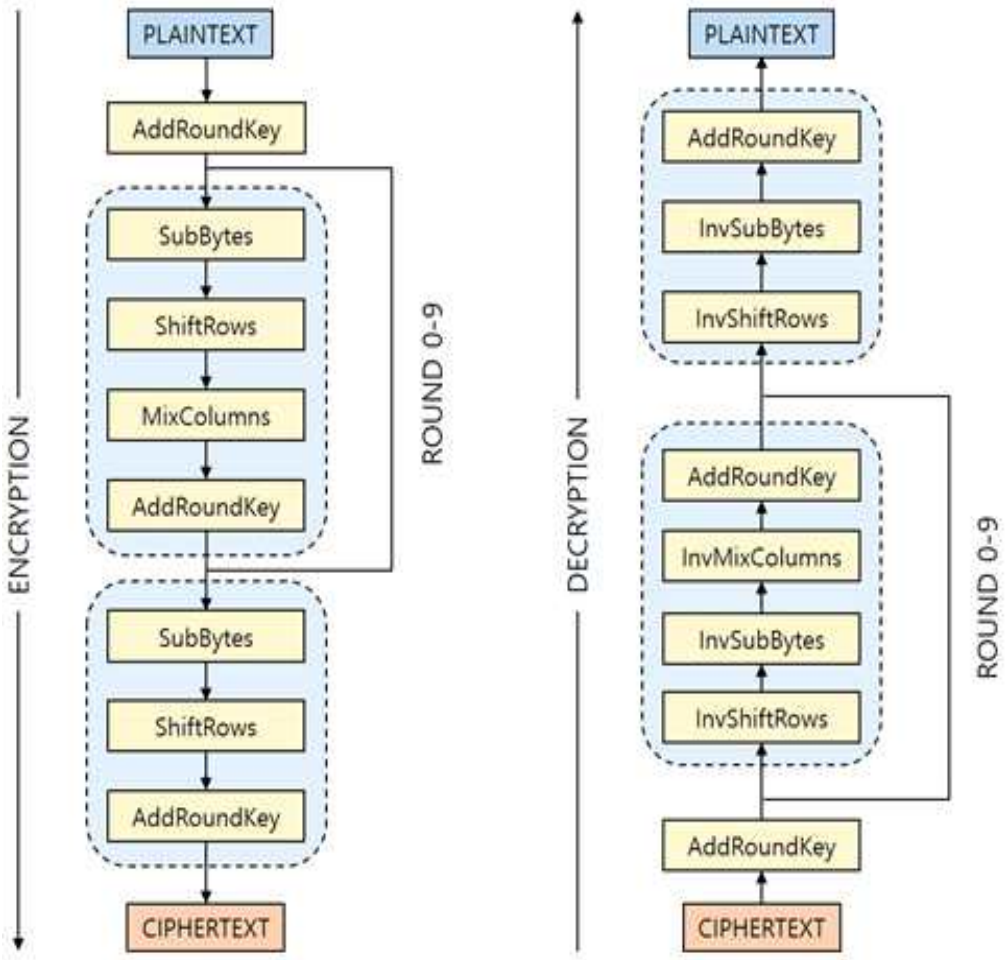
표기법	의미
i	바이트의 순서
j	라운드
S[i]	각 바이트의 입력/출력 값
M[i]	각 바이트의 마스킹 값

Ⅱ. 관련 연구

2.1 AES 암호 알고리즘

2.1.1 AES 암호 알고리즘의 구조

AES 암호 알고리즘은 전 세계에서 가장 널리 사용되는 대칭키 암호 알고리즘으로 고정된 128-bit 블록 크기를 가지고 키의 길이에 따라 128/192/256-bit로 나뉜다. 각각 10/12/14회의 라운드를 거치며 AddRoundKey, SubBytes, ShiftRows, MixColumns 연산을 반복하여 암호문을 생성한다. AddRoundKey 연산은 XOR 연산을 수행하며 연산 당 1개의 바이트에만 영향을 끼치고 다른 바이트에는 영향을 주지 않는다. SubBytes 연산은 미리 저장된 S-Box(Substitution-Box) 테이블을 참조하여 치환 연산을 하여 암호화의 혼돈(Confusion)을 담당한다. 이 또한 AddRoundKey와 같이 대상 바이트를 제외한 다른 바이트에는 영향을 주지 않는다. ShiftRows 연산은 각 행의 순서를 바꾸는 연산으로 암호화의 확산(Diffusion)을 담당한다. 마지막으로 MixColumns 연산의 경우 각 열의 값들을 곱셈 연산을 통해 암호화의 확산을 담당하며 AES 암호 알고리즘의 연산 중 가장 긴 소요시간이 필요한 연산이다. AES 암호 알고리즘의 암호화 및 복호화의 전체적인 동작 구조는 [그림 2-1]에 나타난다.



[그림 2-1] AES 암호 알고리즘

2.1.2 Galois Counter Mode(GCM)의 구조

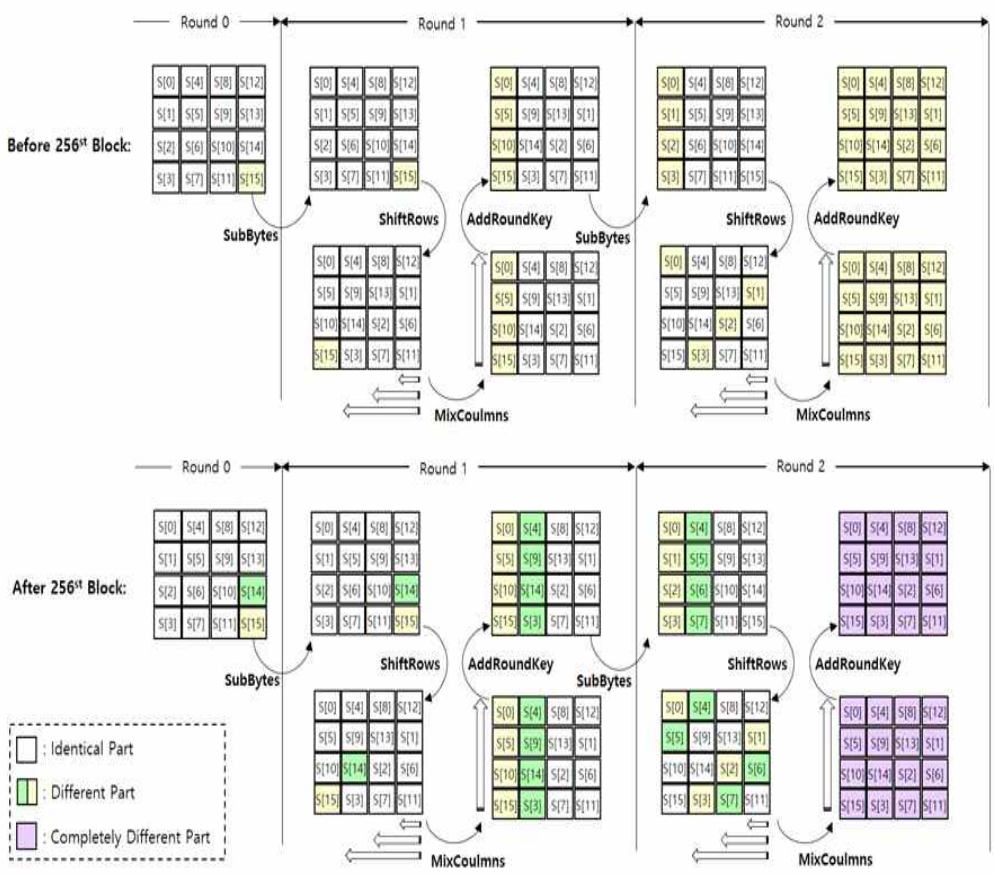
GCM 모드는 최근 주목받고 있는 AEAD를 암호 운영 모드에 적용하여 메세지의 기밀성, 인증성 및 무결성을 모두 보호할 수 있는 암호화 모드이다. GCM은 CTR 모드를 사용하여 암호화를 수행하고 GHASH 함수를 사용하여 MAC(Message Authentication Code)를 생성한다. 따라서 수신자는 암호문을 복호화하기 전에 MAC 값을 비교하여 메세지의 무결성을 인

증 및 송신자 인증까지 할 수 있다. 이러한 방식은 메세지의 특정한 데이터가 암호화되지 않은 경우에도 해당 메세지의 인증성과 무결성을 확인하는 데 사용할 수 있기 때문에 TLS(Transport Layer Security) 같은 네트워크 패킷 암호화에서 가장 많이 쓰이는 모드이다.

2.1.3 FACE: Fast AES-CTR Encryption

2018년 CHES에서 AES-CTR 모드를 Intel 프로세서에서 최적화하여 구현하는 FACE 기법을 발표했다. AES-CTR 모드는 64-bit의 논스(Nonce) 값과 64-bit의 카운터 값으로 이루어진 128-bit의 IV(Initial Vector) 값을 암호화 입력 값으로 사용하는 방식으로 첫 번째 블록의 IV 값과 두 번째 블록의 IV 값의 유일한 차이는 카운터 값으로 사용되는 특정 바이트 밖에 없다는 특징을 가진다. 따라서 IV 값의 일정 부분이 카운터 값의 변경에 의해서만 영향을 받기 때문에 각 블록마다 반복되는 값을 사전 연산 테이블 형태로 저장하고 재사용하여 암호화 연산 시간을 단축한다. FACE 기법은 크게 4 단계로 구성된다. FACE 기법의 전체적인 동작 원리는 [그림 2-2]에 나타난다.

첫 번째 단계는 FACErd0로, 라운드 0의 AddRoundKey 연산의 결과 값을 사전 연산 테이블에 저장한다. AddRoundKey 연산의 경우 입력 값의 다른 바이트에 영향을 미치지 않기 때문에 IV 값에서 카운터 값으로 사용되는 4바이트를 제외한 12 바이트를 미리 계산하여 최적화 할 수 있다. 이렇게 사전 계산된 테이블의 경우 $2^{32}-1$ 번의 암호화 연산에서 업데이트할 필요 없이 반복하여 사용할 수 있다.



[그림 2-2] FACE 기법 개요

FACE의 두 번째 단계는 $FACE_{rd1}$로, 라운드 1의 MixColumns 연산 후 출력 값의 첫 번째 열을 제외한 나머지 세 개의 열을 사전 연산 테이블에 저장한다. 해당 부분은 암호화 과정에서 카운터 값의 변화에 의해 영향을 받지 않는 부분으로 해당 영역은 256번의 암호화 연산에서 반복하여 사용할 수 있고 그 이후에 업데이트 과정이 필요하다.

FACE의 세 번째 단계는 $FACE_{rd1+}$으로, 앞서 $FACE_{rd1}$에서 저장하지 않는 첫 번째 열을 사전 연산 테이블에 저장한다. 해당 부분은 블록의 순서에 따라 변경되는 카운터 값에 의해 영향을 받는 부분이다. 해당 부분은 암호화 과정에서 카운터 값의 증가로 인해 S[10]의 값이 증가할 때 까지

업데이트할 필요가 없기 때문에 약 $2^{40}-1$번의 암호화 연산에서 반복하여 사용할 수 있다.

FACE의 마지막 단계는 $FACE_{rd2}$이다. 라운드 2의 MixColumns 연산이 끝난 이후에는 카운터 값의 변화가 블록 전체에 영향을 끼치기 때문에 이후에는 더 이상 블록마다 반복되는 값을 찾을 수 없다. 하지만 마지막으로 MixColumns 연산 내부에 들어가는 반복된 부분을 사전 테이블에 저장하여 재사용 할 수 있다.

하지만 FACE 기법의 경우 카운터 값의 증가로 인한 사전 연산 테이블의 잦은 업데이트로 부가적인 연산 시간이 필요하고 Timing Attack, SPA 공격과 같은 부채널 공격에 허점을 가진다.

2.2 다항식 곱셈 연산

다항식 곱셈 연산은 암호화 연산에서 많이 사용되는 곱셈 연산으로 두 개의 이진 곱셈과 모듈러 리덕션(Modular Reduction) 연산을 포함한다. 이진 곱셈 연산은 암호화 연산에서 연산 시간이 가장 많이 드는 연산 중 하나로 곱셈 성능을 향상시키기 위한 다양한 연구가 진행 중이다.

첫 번째 연구는 Lopez와 Dahab이 제안한 사전 연산 테이블을 사용하는 기법이다. 앞서 FACE 기법에서 사용했던 사전 연산 테이블과 동일한 원리로 예측 가능한 모든 곱셈의 결과 값을 사전에 연산하여 테이블에 저장하고 입력 값에 따라 해당 테이블을 참조하여 연산 시간을 단축하는 기법이다. 하지만 이러한 기법은 전력 소비 파형 분석을 통한 CPA 공격에 취약성이 밝혀짐으로 최근에는 사용하지 않는다.

두 번째는 여러 개의 블록에 비트 단위의 XOR 연산을 사용하여 이진 곱셈을 수행하는 Block-Comb 기법이다. Block-Comb 기법의 경우 피연산자 비트의 값이 1인 경우에는 XOR 연산을 수행하고 0인 경우에는 연산

을 하지 않는 방식으로 이진 곱셈을 수행한다. 해당 기법은 사전 연산 테이블 기법에 비해 느리지만 최소한의 메모리로 이진 곱셈을 수행할 수 있다. 또한 Seo는 Block-Comb 기법에 Karatsuba 곱셈 알고리즘을 적용시켜 이전에 비해 성능이 향상된 Karatsuba Block-Comb 기법을 제안했다. 하지만 Block-Comb 기법의 경우 피연산자의 값에 따라 연산이 결정되기 때문에 피연산자의 값에 따라 연산 속도가 다르다. 따라서 Timing Attack, SPA 등의 부채널 공격에 취약성을 가진다. 이러한 취약점을 보완하기 위해 항상 일정한 연산 속도를 가질 수 있도록 마스킹 연산을 추가하여 구현한 Masked Block-Comb 기법이 소개되었으나 마스크 값이 0으로 설정되는 경우 제로 레지스터를 사용하여 소비 전력 패턴이 유출되어 CPA 공격에 취약성을 가진다.

가장 최근에 Seo는 이러한 취약점을 보완하기 위해 기존에 사용한 마스킹 기법이 아닌 피연산자 비트가 0인 경우 XOR 결과 값을 위장 레지스터에 저장하는 더미(Dummy) 연산을 추가한 Block-Comb 기법을 제안했다.

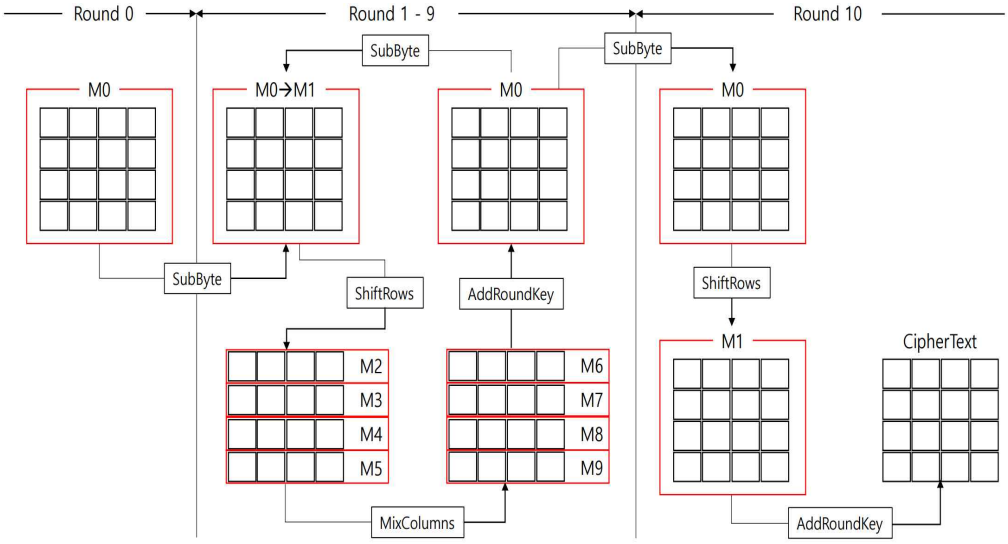
2.3 부채널 공격 및 대응기법

부채널 공격이란 암호 자체의 취약점이 아닌 암호화 연산 과정에서 발생하는 부가적인 정보들을 이용하여 암호의 보안성을 낮추는 공격을 의미한다. 이러한 부채널 공격에는 TA(Timing Attack), 전력 소비 분석 공격 등 다양한 공격 기법이 존재한다. 최근에는 암호화 연산 과정에서 사용되는 전력 소비량을 분석하여 암호문의 키 값을 알아내는 CPA, DPA와 같은 전력 소비 분석 공격에 대한 연구가 활발히 진행되고 있다.

이러한 전력 소비 분석 공격의 경우 소프트웨어로 이루어진 암호 모듈에 공격자가 올바른 소비 전력을 측정할 수 없도록 의미 없는 연산을 추

가하는 마스킹 기법을 통해 방어할 수 있다. 이러한 마스킹 연산은 암호화 과정에서 중간 암호문 값이 노출되지 않도록 XOR 연산을 통해 암호문을 숨기고 최종 라운드에서 실제 암호문이 출력되도록 구현하여 실제 암호 연산에는 영향을 끼치지 않는다.

본 논문에서 사용한 마스킹 기법의 전체적인 동작 과정은 [그림 2-3]에 나타난다.



[그림 2-3] 마스킹 기법 개요

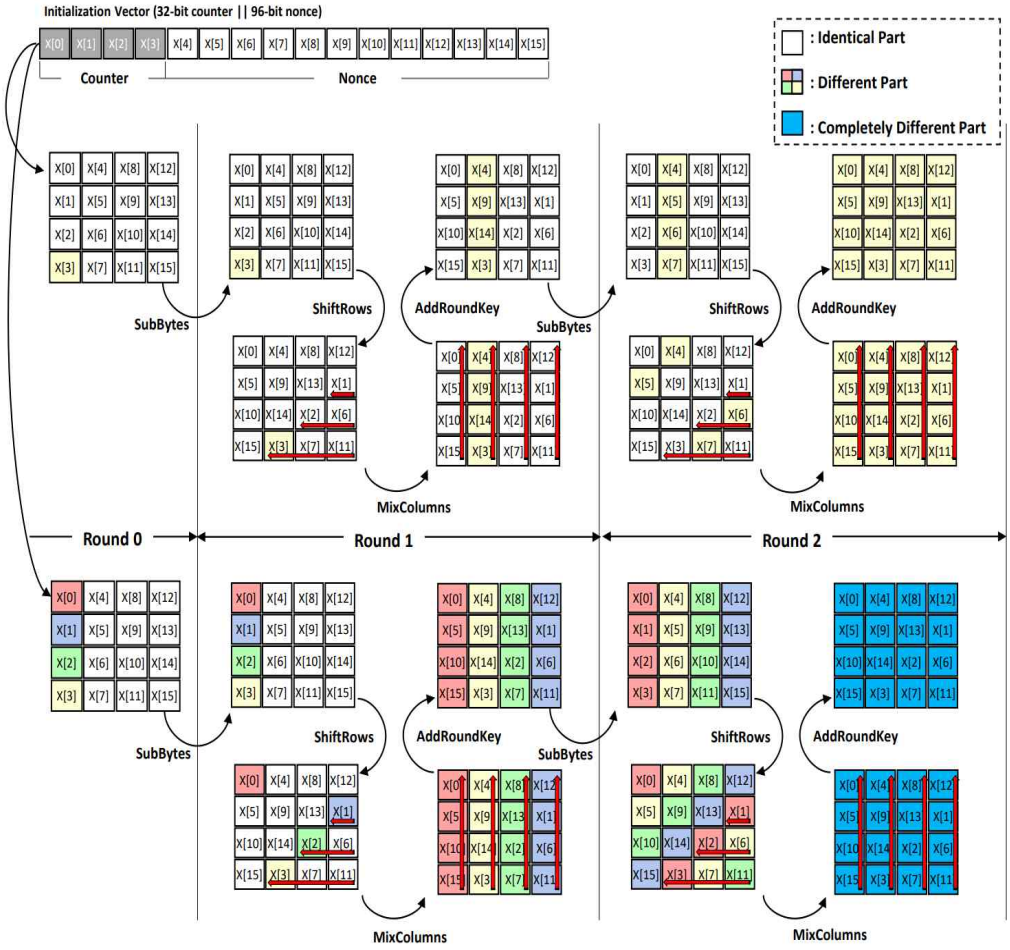
Ⅲ. AES-CTR 모드 최적화

본 장에서는 AES-GCM에서 암호화를 담당하는 AES-CTR 모드의 최적화 구현 기법에 대해서 서술한다. 8-bit 저전력 프로세서에서 사용할 수 없는 단점을 가진 FACE 기법을 경량화하여 저전력 프로세서 환경에서도 사용할 수 있는 경량화 기법 제안 및 성능 향상 기법에 대해 제안한다.

1 절에서는 사전 연산 테이블의 크기를 줄이고 업데이트가 필요 없는 경량화 기법에 대해 서술하고 2 절에서는 FACE 기법에 제안하는 기법을 추가로 구현하여 암호화 속도 향상 최적화 기법에 대해 서술한다. 3 절에서는 전력 소비 분석을 통한 부채널 공격에 대응성을 갖추기 위해 마스킹 연산을 최적화하여 구현하는 기법에 대해 서술한다.

3.1 AES-CTR 모드 경량화 제안 기법

기존의 AES-CTR 모드의 경우 96bit의 논스 값과 32bit의 카운터 값으로 이루어진 IV 값을 암호화의 입력 값으로 사용한다. 이 중 카운터 값이 IV 값의 후반부 4 바이트에 위치한다. 따라서 FACE 기법의 경우 IV 값의 가장 마지막 바이트를 중점으로 두어 해당 바이트 값의 변화에 따라 256번의 암호화 이후에는 사전 연산 테이블을 업데이트해야 한다. 하지만 본 논문에서 제안하는 기법은 IV 값의 상위 4 바이트를 카운터 값으로 설정하고 4 바이트 값 전체를 고려하여 사전 연산 테이블을 생성한다. 따라서 초기화 단계에서 사전 연산 결과들을 테이블에 저장하게 되면 그 이후에는 주기적으로 업데이트를 할 필요가 없다. 제안하는 경량화 기법의 전체적인 과정은 [그림 3-1]에 나타난다. [그림 3-1]의 상위 그림은 256



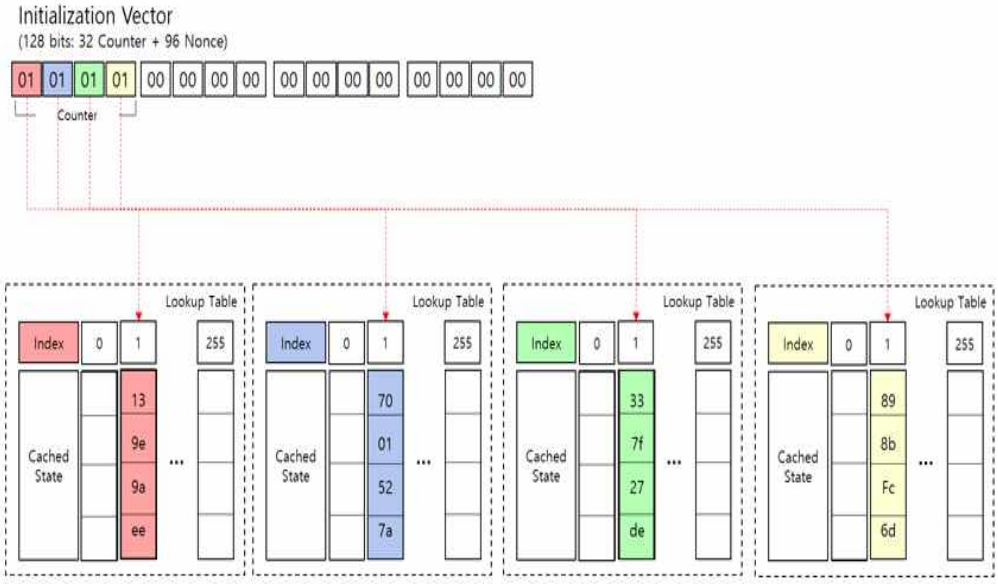
[그림3-1] 제안하는 기법의 동작 방식

번의 암호화 과정에서의 카운터 값의 변화가 암호화 과정에 미치는 영향을 나타내고 하위 그림은 256번의 암호화 이후 블록들의 카운터 값의 변화에 따라 암호화 과정에 미치는 영향을 나타낸다.

라운드 0에서는 평문과 라운드 키 값을 XOR 연산하는 AddRoundKey 연산밖에 없기 때문에 첫 블록과 다음 블록의 차이는 1바이트에 불과하다.

라운드 1부터 SubBytes, ShiftRows, MixColumns, AddRoundKey 연산으로 이루어진 암호화 연산이 라운드가 끝날 때까지 순서대로 반복된다.

SubBytes 연산은 AddRoundKey와 함께 1 바이트의 입력은 1 바이트에만 영향을 주고 주변 다른 바이트에는 영향을 미치지 않는다. ShiftRows 연산은 특정 오프셋(Offset)만큼 각 바이트에 Shift 연산을 수행한다. 따라서 변화된 카운터 값이 다른 열로 이동한다. MixColumns 연산은 ShiftRows 연산으로 각 열로 이동한 1바이트 카운터 값들이 총 4바이트의 같은 열의 다른 값들과 곱셈 연산을 거치면서 처음 블록과 N 번째 블록들 사이에 더 이상 공통된 값이 없어진 것을 알 수 있다. 하지만 [그림 3-2]의 하단을 보면 카운터 값으로 설정했던 S[0]의 값이 MixColumns 연산을 거치면서 첫 번째 열(S[0], S[5], S[10], S[15])에만 영향을 미치는 것을 알 수 있고 이는 각 열에 분포된 S[1], S[2], S[3]에도 동일하게 적용된다. 마지막 AddRoundKey 연산의 경우 다른 바이트에 영향을 미치지 않기 때문에 각 열의 결과 값은 1 바이트의 카운터 값에 의존하여 0부터 255까지 256개의 케이스에 유효한 사전 연산 테이블을 제작할 수 있다.



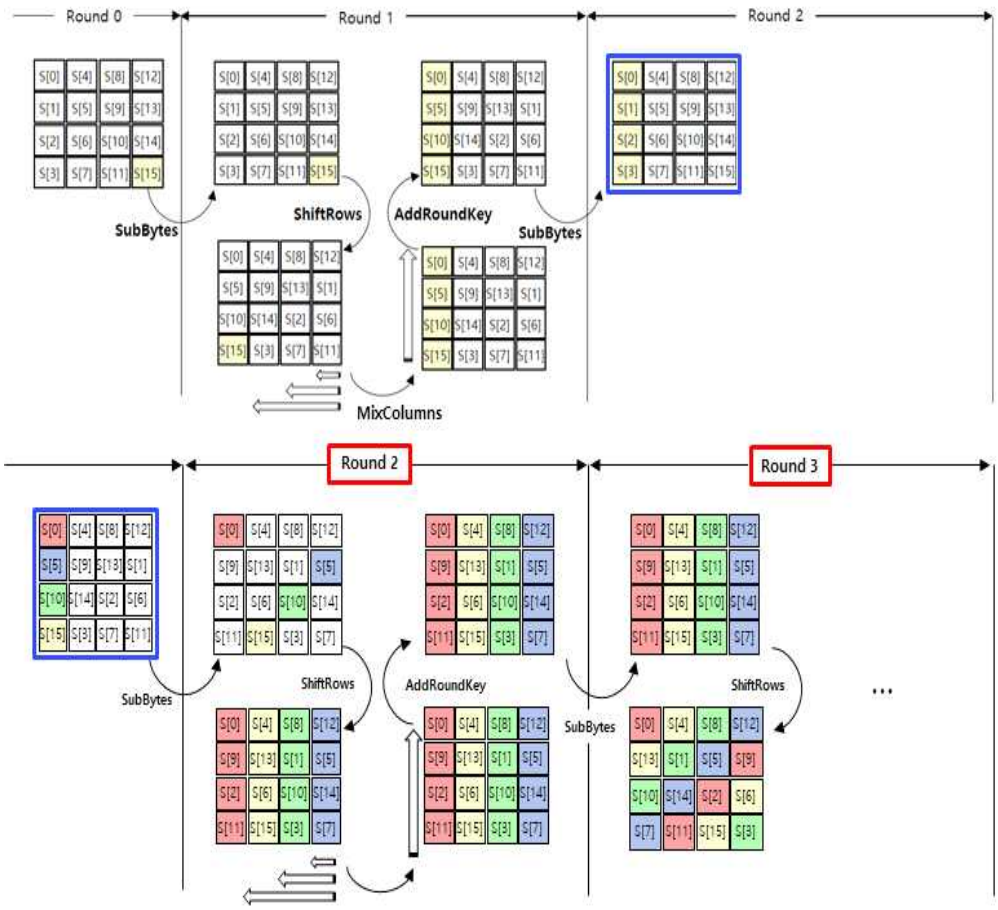
[그림3-2] 제안하는 기법의 사전 연산 테이블 구조

라운드 2에서는 라운드 1을 거친 암호문이 SubBytes, ShiftRows 연산을 거치는 동안은 값이 다른 바이트들과 뒤섞이지 않기 때문에 사전 연산 테이블을 제작할 수 있다. 따라서 카운터 값으로 설정된 S[0], S[1], S[2], S[3]의 값에 의해 라운드 2의 MixColumns 연산 직전까지의 연산 결과 값을 사전에 계산하여 테이블에 저장할 수 있다. 예를 들어 S[0]의 경우 S[0]의 값이 변할 때 4 바이트로 이루어진 첫 번째 열의 값들이 영향을 받기 때문에 S[0]가 가질 수 있는 256개의 케이스에 맞춰 첫 번째 열의 사전 연산 테이블을 제작하면 1KB의 메모리 크기로 업데이트가 필요 없는 사전 연산 테이블을 제작할 수 있다. 이는 다른 카운터 값(S[1], S[2], S[3])에도 동일하게 적용되며 그 결과 총 4KB의 메모리 공간으로 약 2개의 라운드 연산을 단축하여 연산 시간을 최적화 할 수 있다. 해당 기법은 [그림 3-3]에 자세히 나타난다.

제안하는 기법은 Intel 프로세서에서 최적화된 FACE 기법과는 다르게 최소화된 성능을 사용하는 8-bit 저전력 프로세서에서도 사용할 수 있게 사전 연산 테이블 크기가 줄었고 업데이트가 필요 없기 때문에 SRAM이 아닌 Flash 메모리(PROGRAM MEMORY)에 사전 연산 테이블 값을 저장할 수 있어서 암호화 이외의 다른 연산을 수행하는데 메모리 공간을 사용할 수 있는 장점을 가진다.

3.2 향상된 성능의 FACE 제안 기법

제안하는 기법은 기존 AES-CTR 모드에서 사용하는 방식과 다르게 IV 값의 카운터 값을 전위에 위치시킴으로써 기존 연구인 FACE와 결합하여 사용할 수 있다. FACE 기법의 경우 최하위 바이트의 값의 변화에 따라 사전 연산 테이블 값이 결정되는 반면에 제안하는 기법은 카운터 값이 전위에 위치하기 때문에 FACE 기법으로 라운드 1까지 진행한 이후에 제



[그림3-3] 제안하는 성능 향상된 FACE 기법

안하는 기법을 적용하면 변화된 첫 번째 열의 값에 상관없이 라운드 3까지 확장하여 적용시킬 수 있다.

해당 기법은 [그림 3-4]에 나타난다. [그림 3-4]에 상위 그림을 보면 FACE 기법인 것을 알 수 있고 라운드 1 이후 하단 그림에서 제안하는 기법을 추가로 적용하여 라운드 3의 MixColumns 연산 직전까지 확장시켜 적용한 것을 볼 수 있다.

3.3 AES-CTR 모드 최적화 구현 기법

제안하는 기법에서 사용할 사전 연산 테이블은 암호화 연산 전 초기화 단계에서 사전 연산되어 저장된다. 대부분의 8-bit AVR 프로세서는 2~4KB 정도의 SRAM 공간을 가지기 때문에 사전 연산 테이블을 저장하기에는 매우 제한적이다. 따라서 제안하는 기법은 사전 연산 테이블을 저장하기 위해 PROGRAM MEMORY를 사용한다. 각 사전 연산 테이블은 32-bit의 결과 값을 출력하기 때문에 8-bit의 입력 값 4개을 합쳐서 오프셋(Offset)으로 사용함으로써 32-bit 입력 값으로 확장하여 사용한다. 그리고 입력 오프셋을 이용하여 각 사전 연산 테이블의 시작 주소를 기준으로 사전 연산 값을 참조한다.

AES 암호 알고리즘은 SubBytes 연산중에 치환 연산이 수행된다. 작업 중 치환 연산을 해야 하는 256 바이트 값을 미리 SBOX라는 테이블로 메모리에 저장해 연산 시간을 단축한다. SubBytes 연산은 입력 값을 기반으로 SBOX 테이블에 저장된 사전 연산 값들과 치환 연산을 통해 결과 값으로 출력하기 때문에 Z 포인터(R30, R31)를 사용하여 SBOX에 저장된 값을 로드해야 한다. 이 때 R30 레지스터에는 SBOX의 하위 메모리 주소가 저장되고 R31 레지스터에는 SBOX의 상위 메모리 주소가 저장되기 때문에 인덱스 값으로 사용될 입력 값을 ADD 명령어를 이용하여 R30 레지스터 값에 더해서 올바른 메모리 주소를 설정해야 한다. 하지만 R30 레지스터에 저장된 SBOX 메모리의 하위 주소에 입력 값을 ADD 연산할 때 오버플로우(Overflow)로 인한 캐리(Carry) 값이 발생할 수 있으므로 이런 경우 ADC 명령어를 이용하여 캐리 값이 발생한 경우 R31 레지스터의 값을 증가시켜야한다.

그러나 ADD 연산에서 캐리 값이 발생하지 않은 경우에 ADC 명령어는 0을 더하는 의미 없는 연산일 뿐만 아니라 SBOX 메모리를 참조할 때마다 2번의 연산(ADD, ADC)이 수행되기 때문에 큰 오버헤드가 발생한다. 이 문제

를 해결하기 위해 SBOX 테이블의 메모리 공간이 256 바이트라는 것을 활용할 수 있다. SBOX 테이블의 시작 주소의 하위 주소를 0x00로 맞춰서 설정한다면 입력 값의 최대가 0xff이기 때문에 메모리 참조를 위한 인덱스 연산을 하는 과정에서 캐리 값이 발생하지 않는다. 따라서 테이블 참조 연산마다 ADC 연산을 생략할 수 있고 추가적으로 ADD 연산이 아닌 R30 레지스터에 입력 값을 바로 MOV 명령어로 로드할 수 있다. 256 바이트 이하의 크기의 테이블 참조 연산을 하는 경우 동일한 기법을 사용할 수 있다.

LD와 ST 명령어는 연속된 메모리에 저장된 키 값과 라운드 키 값을 로드하거나 저장하는 데 사용된다. AVR 프로세서의 경우 X(R26, R27), Y(R28, R29), Z(R30, R31) 포인터를 사용하여 16비트 주소에 접근한다. 따라서 반복 메모리나 주변 메모리에 접근할 때는 포인터를 구성하는 레지스터에서 ADD와 ADC 명령어를 사용하여 접근할 메모리 주소를 설정해야 한다. 이 경우 위와 동일하게 상당한 오버헤드를 가진 캐리 값 연산이 포함되어야 한다. 따라서 연속되는 메모리 주소에 접근하거나 근처의 주소에 접근하는 경우 추가적인 명령어 없이 주소를 바로 계산할 수 있는 AVR이 제공하는 LDD, STD등이 명령어를 사용하여 구현함으로써 시간을 단축할 수 있다.

3.3 마스킹 연산 최적화 구현 기법

마스킹 연산은 암호화 연산 결과 자체에는 어떠한 영향도 주지 않고 불필요한 연산을 추가하여 공격자가 전력 사용량을 정확하게 측정하는 것을 방지하는 역할을 한다. 따라서 마스킹을 적용시킨 암호화 연산 시간은 일반적인 암호화 연산 시간보다 길어질 수밖에 없다. 본 논문에서는 이러한 불이익을 최소화하기 위해 두 가지 방법을 제안한다.

Masking operation optimization

```
1: mov HI, ROUND
2: lsl HI
3: lsl HI

4: eor MASK6, MASK0
5: eor MASK7, MASK0
6: eor MASK8, MASK0
7: eor MASK9, MASK0

8: 1:
9: ld r0, Z
10: eor r0, MASK6
11: st Z+, r0

12: ld r0, Z
13: eor r0, MASK7
14: st Z+, r0

15: ld r0, Z
16: eor r0, MASK8
17: st Z+, r0

18: ld r0, Z
19: eor r0, MASK9
20: st Z+, r0

21: dec HI
22: brne 1b

23: ldi HI, 4

24: 2:
25: ld r0, Z
26: eor r0, MASK1
27: st Z+, r0

28: ld r0, Z
29: eor r0, MASK1
30: st Z+, r0

31: ld r0, Z
32: eor r0, MASK1
33: st Z+, r0

34: ld r0, Z
35: eor r0, MASK1
36: st Z+, r0

37: dec HI
38: brne 2b
```

[알고리즘 3-1] 최적화된 마스킹 연산 코드

첫 번째 방법은 고정키 암호화 방식을 사용하여 암호화 작업을 수행하기 전에 라운드 0부터 라운드 9까지의 10세트의 라운드 키 값에 M[0] 값을 XOR 연산한다. 그런 다음 첫 번째 행에 M[6], 두 번째 행에 M[7], 세 번째 행에 M[8], 마지막 행에 M[9]로 XOR 연산을 수행한다. 마지막 라운드인 10라운드의 경우에는 M[6], M[7], M[8], M[9]에 대한 XOR 연산을 하지 않고 모든 값에 M1 값에 XOR 연산을 적용한다. 해당 연산을 미리 실시하면 각 라운드에서 반복되는 XOR 연산을 최소화하고, 마스킹 값이 저장되는 메모리를 각 라운드마다 로드할 필요가 없다. [알고리즘 3-1]의 4행에서 M[0]는 M[6], M[7], M[8], M[9]와 XOR 연산되는 것을 볼 수 있다. 그리고 Lable 1에서 M[6], M[7], M[8], M[9]를 라운드 0부터 라운드 9까지 총 10개의 키 세트에 대해 행별로 XOR 연산한다. 이처럼 각 마스크 값에 M0

값을 미리 XOR 연산하면 약 160개의 XOR 명령어를 줄일 수 있다. Lable 2에서도 동일하게 마지막 라운드 키 값에 대해 M[1] 값을 XOR 연산하는 것을 볼 수 있다.

두 번째는, SubBytes 연산은 테이블로 저장된 SBOX 메모리에 접근해야하기 때문에 상대적으로 연산 시간이이 많이 소요된다. 또한 SubBytes 연산을 수행한 후에는 M[0]를 M[1]으로 교체해야 하기 때문에 매 라운드마다 메모리 로드 연산과 XOR 연산을 실행할 때 오버헤드가 발생한다. 오버헤드를 최소화하기 위해 Masked-SBOX 테이블을 사전 계산하고 SubBytes 작업에서 참조한다. 구현에 대한 자세한 설명은 [알고리즘 3-2]에 제시되어 있다.

Generating Masked SBOX

```
1: ldi r31, hi8(MSBOX)
2: ldi r29, hi8(SBOX)
3: ldi xREDUCER, 0x1b

4: 1:
5: mov r28, r0
6: ld r26, Y
7: eor r26, T1
8: mov r30, r0
9: eor r30, T0

10: st Z, r26

11: inc r0
12: brne 1b
```

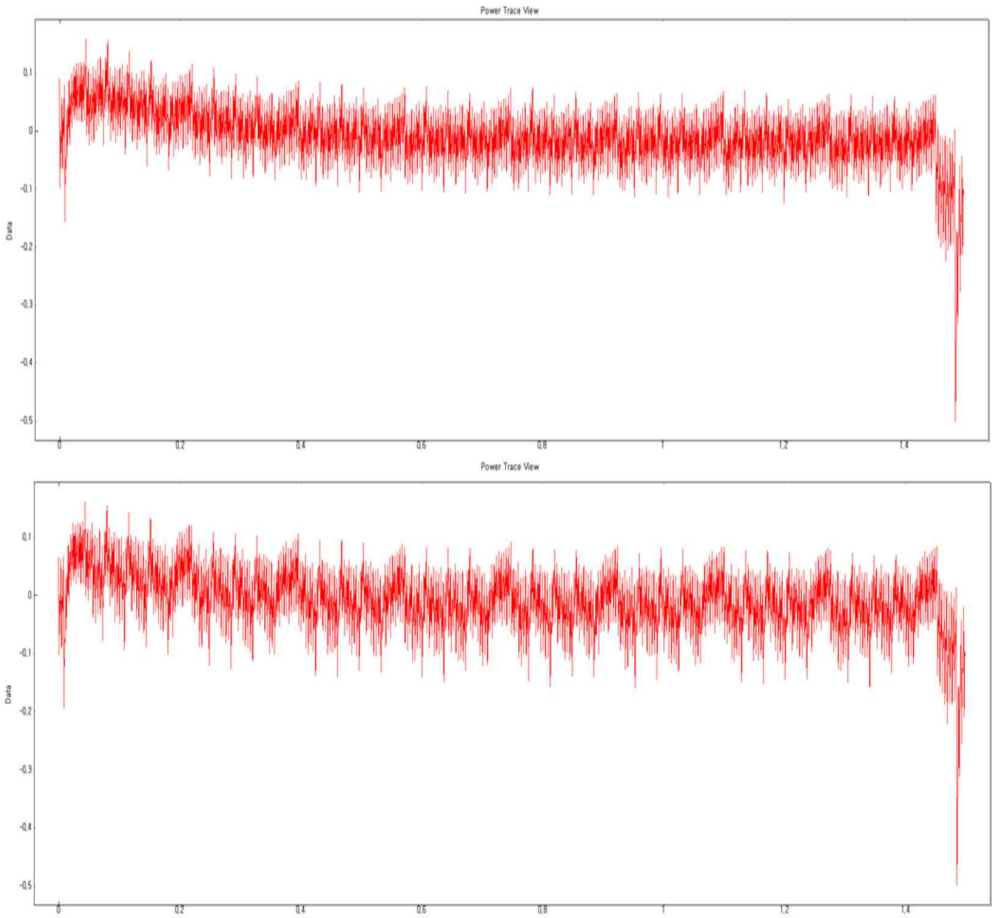
[알고리즘 3-2] SBOX 테이블 사전 마스킹 코드

Ⅳ. GHASH 함수 최적화

본 장에서는 2019년에 Seo와 Kim이 발표한 128-bit 이진 곱셈 연산 최적화 연구 결과와 비교하여 제안하는 기법에 대해 서술한다. 첫 번째로 이전 연구는 8개의 위장 레지스터를 활용하여 GHASH 함수의 128-bit 이진 곱셈을 계산하였다. 그러나 제안하는 기법은 CPA, SPA, TA와 같은 공격에 대한 보안상 손실 없이 하나의 위장 레지스터만을 활용한다. 또한, 이진 곱셈 연산에서 레지스터를 사용을 최소화 하도록 구현된다. 따라서 레지스터 최적화로 확보된 레지스터를 이용하여 Karatsuba 곱셈 알고리즘에서 이전 방식보다 총 8개의 레지스터를 추가로 활용한다. 둘째, GHASH 함수 연산에서 반복되는 값을 통해 Karatsuba 곱셈 과정에서 피연산자의 일부를 사전 연산할 수 있다. 사전 계산된 값은 한 번 생성되고 여러 번 호출되므로 연산 시간을 단축시킬 수 있다. 마지막으로, Shay와 Kounavis가 제안한 최적화된 모듈러 리덕션 기법을 적용시켜 Seo와 Kim이 제안한 기법에 비해 모듈러 리덕션 연산 시간을 단축한다. 제안하는 기법은 8-bit AVR 프로세서에 최적화되어 있다.

4.1 128-bit 이진 곱셈 연산 최적화 제안 기법

Seo와 Kim이 제안한 Block Comb 기법을 이용한 연구는 8개의 위장 레지스터를 이용하여 실제 연산과 동일한 명령어와 명령어 수로 이루어진 위장 연산을 수행하여 SPA, TA, CPA 공격을 방지한다. 따라서 8개의 위장 레지스터에 저장된 값들은 실제로는 사용되지 않는다. 하지만 공격자가 전력 소비 파형을 분석할 때 동일한 명령어로 이루어진 실제 연산과 위장 연산의 차이점을 찾을 수 없기 때문에 실제 연산과는 상관없이 의미 없는 데이터에 대



[그림 4-1] 8개의 위장 레지스터를 사용한 AES 전력 소비량 파형(上)
1개의 위장 레지스터를 사용한 AES 전력 소비량 파형(下)

한 연산을 수행한다.

하지만 한정된 자원을 가지고 있는 저전력 프로세서에서 8개의 레지스터는 상당히 중요한 자원이기 때문에 이러한 위장 연산은 곱셈 연산에 상당히 큰 오버헤드를 야기한다. 따라서 본 논문에서는 이러한 문제점을 해결하기 위해 8개의 위장 레지스터를 1개의 위장 레지스터로 대체한다. [그림 4-1]은 32-bit 이진 곱셈 연산의 소비 전력 파형을 나타낸다. 상위 파형은 8개의 위장 레지스터를 사용하는 연산에 대한 전력 소비 파형이고 하위 파형은

Proposed method for 32-bit multiplication.

Input: : 32-bit multiplicand A ($R_{19}, \ldots, R_{16}$), 32-bit multiplier B ($R_{23}, \ldots, R_{20}$), Garbage result (R_{24}), and real result ($R_{15}, \ldots, R_8$).
Output: : Result C (64-bit) = $A \times B$.

```
...
1: R25 ← 0x06
2: for l = 7 to 0 do
3:   for m = 3 to 0 do
4:     if the l-th bit of R16+m==1 then
5:       R0 ← R0 + R25
6:       for k = 0 to 3 do
7:         R8+m+k ← R8+m+k ⊕ R20+k
8:       end for
9:     else
10:      for k = 0 to 3 do
11:        R24 ← R24 ⊕ R20+k
12:      end for
13:    end if
14:  end for
15:  (R15, ..., R8) ← (R15, ..., R8) ≪ 1
16: end for
...
```

[알고리즘 4-1] 제안하는 32-bit 이진 곱셈 연산

1개의 위장 레지스터를 사용하는 전력 소비 파형이다. 이 때 상위 파형과 하위 파형 모두 동일하게 규칙적인 패턴을 가지고 있는 것을 알 수 있다. 따라서 [알고리즘 4-1]의 11번째 명령어에서 보이는 것처럼 1개의 위장 레지스터(R24)로 8개의 위장 레지스터를 사용하는 것을 대체하는 것을

볼 수 있다.

또한 제안하는 기법은 Liu가 제안했던 128-bit 이진 곱셈 연산 최적화 기법을 적용하여 이전 연구에 비해 성능을 향상시켰다. Seo와 Kim의 제안 기법은 64-bit 누산기(Accumulator)에 Shift 연산하는 방식을 사용하지 않고 64-bit 곱셈 연산에서 40-bit로 이루어진 피연산자 (A)를 Shift 연산하는 방식을 제안했다. 이러한 방식은 32-bit 곱셈 연산 당 29개의 Shift 명령어를 줄일 수 있다. 하지만 이러한 방식은 Liu가 제안했던 기법에 비해서 큰 성능 향상을 보이지 않는다. 왜냐하면 Seo와 Kim의 기법에 따르면 Liu의 기법에 비해 Shift 명령어가 줄긴 하지만 XOR 명령어는 1-bit 연산마다 1회씩 더 작동하므로 40-bit 피연산자 (A)를 계산하는 동안 32개의 XOR 명령어가 추가된다. 따라서 전체적인 연산 시간으로 볼 때 큰 차이를 보이지 않는다. 게다가 Seo와 Kim의 기법을 사용하면 40-bit 피연산자 (A)를 저장하기 위해 5개의 레지스터를 사용하기 때문에 Liu의 기법에 비해 레지스터 1개가 더 필요하다. 따라서 Liu가 제안한 곱셈 기법을 이용하면 비슷한 연산 시간에 1게의 레지스터를 더 확보할 수 있다. 이렇게 확보한 레지스터 8개는 128-bit 이진 곱셈 연산에서 최적화를 위해 사용되는 Karatsuba 곱셈 알고리즘에서 사용할 수 있다. 이와 같은 레지스터 최적화 기법은 자원 사용이 최소화돼야 하는 다른 저전력 프로세서들에서도 사용될 수 있다.

4.2 Karatsuba 곱셈 연산 최적화 제안 기법

Karatsuba 곱셈 알고리즘은 성능이 좋은 곱셈 방식으로 알려져 있으며, 제안하는 기법도 성능 향상을 위해 Karatsuba 곱셈 알고리즘을 사용한다.

Karatsuba 곱셈 알고리즘의 전체적인 동작 방식은 [알고리즘 4-2]에 나타난다. 첫 번째로 두 개의 64-bit 피연산자들의 하위 32-bit(A[3]~A[0], B[3]~B[0])에 대한 곱셈을 수행하여 64-bit의 결과 값 (L)을 구한다.

64-bit Karatsuba Block Comb

Require: 64-bit wise operands A and B

Ensure: Result $C \leftarrow A \cdot B$

1: $L \leftarrow A[3,0] \times B[3,0]$

2: $H \leftarrow A[7,4] \times B[7,4]$

3: $M \leftarrow (A[7,4] \oplus A[3,0]) \times (B[7,4] \oplus B[3,0])$

4: $M \leftarrow M \oplus L \oplus H$

5: $C \leftarrow (H \ll 64) \oplus (M \ll 32) \oplus L$

6: **return** C

[알고리즘 4-2] Karatsuba 곱셈 동작 원리

두 번째로 64-bit 피연산자들의 상위 32-bit(A[7]~A[4], B[7]~B[4])에 대한 곱셈을 수행하여 64-bit의 결과 값 (H)을 구한다. 세 번째로 64-bit의 피연사자 A, B의 상위 값과 하위 값들을 XOR 연산 하고 그 결과를 곱한 결과 값(M)을 구한다. 마지막으로 앞에서 연산했던 결과 값의 자릿수를 맞추기 위하여 (H) 값과 (M)에 (H << 64)와 (M << 32)의 Shift 연산을 한 이후에 모든 값을 XOR 하면 올바른 곱셈 결과를 얻을 수 있다. 해당 방법을 이용하면 기존 곱셈 기법에 비해 총 연산 수가 줄어들어 성능을 향상 시킬 수 있다.

Karatsuba 곱셈 알고리즘은 각 단계마다 나오는 중간 결과 값들을 버퍼 메모리에 저장해야한다. Seo와 Kim이 제안하는 기법은 32개의 레지스터를 전부 사용하기 때문에 이러한 중간 결과 값을 저장하기 위해서는 스택(Stack) 메모리 영역을 사용해야 한다. 하지만 스택 메모리를 사용해서 값을 저장 및 로드하는 경우 범용 레지스터에 비해서 오버헤드가 크게 발생한다. 왜냐하면 레지스터에서 값을 로드하는 명령어는 1 클록 사이클(Clock Cycle)이 소요되는 반면에 스택 메모리에서 값을 로드하는 명령어의 경우 2 Clock Cycle이 소요되기 때문이다. 따라서 본 논문에서 제안하는 기법은 앞서 레지스터 최적

Proposed implementation of 1-level Karatsuba Block-Comb in source code level.

```
1: ROUND32
2: STD Z+0, C0
3: STD Z+1, C1
4: STD Z+2, C2
5: STD Z+3, C3
6: EOR C0, C4
7: EOR C1, C5
8: EOR C2, C6
9: EOR C3, C7
10: MOVW K0, C0
11: MOVW K2, C2
12: MOVW K4, C4
13: MOVW K6, C6
14: ROUND32
15: STD Z+12, C4
16: STD Z+13, C5
17: STD Z+14, C6
18: STD Z+15, C7
19: EOR K0, C0
20: EOR K1, C1
21: EOR K2, C2
22: EOR K3, C3
23: EOR C0, C4
24: EOR C1, C5
25: EOR C2, C6
26: EOR C3, C7
27: EOR K4, C0
28: EOR K5, C1
29: EOR K6, C2
30: EOR K7, C3
31: ROUND32
32: EOR C0, K0
33: EOR C1, K1
34: EOR C2, K2
35: EOR C3, K3
36: EOR C4, K4
37: EOR C5, K5
38: EOR C6, K6
39: EOR C7, K7
40: STD Z+4, C0
41: STD Z+5, C1
42: STD Z+6, C2
43: STD Z+7, C3
44: STD Z+8, C4
45: STD Z+9, C5
46: STD Z+10, C6
47: STD Z+11, C7
```

[알고리즘 4-2] Karatsuba 곱셈 코드

화 기법을 이용하여 확보한 8개의 레지스터에 Karatsuba 알고리즘의 중간 결과 값들을 저장함으로써 성능을 향상시킬 수 있다(레지스터가 한정적인 경우 스택 메모리 사용 가능). [알고리즘 4-3]에서 제안된 구현은 (H), (L), (M) 등의 중간 결과 값들을 스택 메모리에 저장하지 않고 총 8개의 레지스터(K0~K7)에 저장하여 사용하는 것을 볼 수 있다.

128-bit 결과 값 (C)운 8-bit 저전력 프로세서이기 때문에 8-bit 단위로 레지스터에 나누어 저장할 때 (C[0], C[1], ..., C[15])로 총 16개의 레지스터로 나눠진다. 최종 결과 값을 저장할 때 최상위 32-bit, 최하위 32-bit의 경우 중가 결과 값과 추가적인 연산이 필요 없기 때문에 C[0]~C[3]까지는 상위 32-bit 연산 결과를 저장하고 C[12]~C[15]까지는 하위 32-bit 연산 결과를 저장한다. 하지만 나머지 C[4]~C[11]까지 64-bit의 값은 중간 결과 값인 M[0]~M[7]과 XOR 연산을 해야 한다. 예를 들어 C[4]에 들어갈 최종 결과 값은 C[4] ⊕ M[0] ⊕ C[0] ⊕ C[8]이다. 이와 같이 C[4]~C[11]까지 XOR 연산되어야 하는 중간 값들을 저장하기 위해 8개의 레지스터를 추가로 이용하여 버퍼로 사용하면 스택 메모리를 버퍼로 사용하는 기존의 기법에 비해 큰 성능 향상을 이룰 수 있다.

4.3 모듈러 리덕션 최적화 제안 기법

모듈러 리덕션 연산을 최적화하기 위해, 제안하는 기법에 Gueron과 Kounavis가 제안한 고속 모듈러 리덕션 알고리즘을 적용했다. 해당 알고리즘은 64-bit 환경에서 기약 다항식 ($x^{128} + x^7 + x^2 + x + 1$)에서 모듈러 리덕션을 최적화한다. 본 논문에서는 해당 알고리즘을 8비트 AVR 플랫폼에 최적화시켜 적용한다. [알고리즘 4-4]에서는 기존 알고리즘의 방식과는 다르게 입력 값들을 반전하여 K0~K31 레지스터에 저장하기 때문에 기존 알고리즘과 순서가 거꾸로 된 것을 볼 수 있다. 제안하는 알고리즘은 8-bit AVR 프로세서에 최적화된 구현 기법으로 8-bit 환경에서 사용할 수 있도록 Shift 연산이 추가된 것을 볼 수 있다.

Proposed method for fast modular reduction.

Input: : 256-bit data $K[0], ..., K[31]$, where $K[0 \sim 31]$ are 8-bit long each.

Output: : $K[16], ..., K[31]$ (reduced result).

1: $A \leftarrow (K[31] \,\&\, 0b1) \ll 6$
2: $B \leftarrow (K[31] \,\&\, 0b10) \ll 5$
3: $C \leftarrow (K[31] \,\&\, 0b1111111)$

4: $K[16] \leftarrow K[16] \oplus ((A \oplus B \oplus C) \ll 1)$

5: $K[8] \leftarrow K[8] \oplus K[24] \oplus ((K[23] \ll 7) \mid (K[24] \gg 1)) \oplus ((K[23] \ll 6) \mid ((K[24] \gg 2)) \oplus ((K[23] \ll 1) \mid (K[24] \gg 7))$

6: $K[0] \leftarrow K[0] \oplus K[16] \oplus (K[16] \gg 2) \oplus (K[16] \gg 7)$

7: **for** $l = 1$ to 7 **do**

8: $K[i+8] \leftarrow K[i+8] \oplus K[i+24] \oplus ((K[i+23] \ll 7) \mid (K[i+24] \gg 1)) \oplus ((K[i+23] \ll 6) \mid (K[i+24] \gg 2)) \oplus ((K[i+23] \ll 1) \mid (K[i+24] \gg 7))$

9: $K[i] \leftarrow K[i] \oplus K[i+16] \oplus ((K[i+15] \ll 7) \mid (K[i+16] \gg 1)) \oplus ((K[i+15] \ll 6) \mid (K[i+16] \gg 2)) \oplus ((K[i+15] \ll 1) \mid (K[i+16] \gg 7))$

10: **end for**

[알고리즘 4-3] 최적화된 모듈러 리덕션

Ⅴ. 성능 평가

본 논문은 저전력 프로세서인 8-bit AVR 프로세서에 최적화된 AES-GCM 암호 최적화 구현을 제안한다. 타겟 보드로는 ATmega328 프로세서가 탑제된 Arduino UNO 보드를 사용한다. [표 5-1]의 내용에는 Arduino UNO 보드에 탑재된 ATmega328 프로세서의 스펙을 기제한다. 또한 Atmega328 보드는 연산을 위한 범용 레지스터 32개(R0~R31)를 가지고 있고 131개의 명령어를 기본적으로 제공한다. 제안 기법의 구현물은 Arduino IDE와 Atmel Studio 7 환경에서 제작됐고 최적화 레벨 -Os 옵션에서 컴파일 되었으며 암호 연산 시간은 클록 사이클로 측정한다. 또한 전력 사용량을 분석하는 부채널 공격의 대응성을 증명하기 위해 구현물에 Chipwisperer-Lite(CW1173)를 이용하여 전력 소비량을 측정하였고 측정된 전력 소비량을 기준으로 Python 3를 이용하여 CPA, DPA, SPA 공격을 시도하여 부채널 대응성을 증명한다. 본 장에서 서술된 모든 결과는 8-bit AVR 프로세서에서 측정된 결과이다.

[표 5-1] Atmega328 상세 스펙

Atmega328	**Spec**
Clock Speed	16MHz
Operating Voltage	5V
Maximum Supply Voltage	20V
SRAM	2KB
EEPROM	1KB
Flash Memory	32KB

5.1 AES-CTR 모드 최적화 기법 성능 평가

[표 5-2]은 기존에 발표된 AES 최적화 구현 기법들의 성능과 제안 기법들의 성능을 바이트 당 클록 사이클을 기준으로 비교한다. 기존 기법들은 AES-ECB(Electronic CodeBook) 모드를 지원하는 반면 제안 기법은 AES-CTR 모드를 지원한다. 하지만 ECB 모드와 CTR 모드는 AES 암호 알고리즘의 동일한 암호화 연산을 거치기 때문에 비교하는데 무리가 없다. [표 5-2]에 나타난 Dinu의 제안 기법은 코드 크기를 최적화하여 구현한 결과 값이고 Otte의 제안 기법은 암호화 연산 속도를 최적화하여 구현한 결과 값이다. 제안하는 기법은 이전 기법들에 비해 바이트당 최대 18회의 클록 사이클을 단축시켰다. 따라서 속도 최적화 기법인 Otte의 기법의 클록 사이클과 비교했을 때 128-bit, 192-bit, 256-bit의 보안 단계에서 각각 11.5%, 9.6%, 8.3% 빠르다. 또한 이전 연구인 FACE 기법에 제안 기법을 적용시키면 바이트 당 34회의 클록 사이클을 단축할 수 있으며 이는 Otte의 기법에 비해 21.5%, 18.1%, 15.6%의 성능 향상을 보인다.

[표 5-2] AES 최적화 구현 기법 비교

Security Level	Dinu et al.	Otte et al.	This Work	FACE + This Work
128-bit	177	156	138	122
192-bit	N/A	186	168	153
256-bit	N/A	217	199	183

[표 5-3]는 제안하는 기법과 이전 연구인 FACE 기법의 기능을 비교한다. 우선 제안 기법의 경우 초기화 단계 이후에는 사전 연산 테이블을 업데이트 할 필요가 없다. 그 결과 암호화 연산 과정에서 부가적인 연산이 없기 때문에 평문 블록의 개수에 상관없이 일정한 연산 시간을 가진다. 따라서 TA, SPA 와 같은 부채널 공격에 대응성을 가질 수 있다. 두 번째로 FACE 기법의 경우 저전력 프로세서에서 사용하기에는 사전 연산 테이블의 크기가 크고 주기적인 업데이트가 필요하기 때문에 저전력 프로세서에서는 사용할 수 없지만 제안하는 기법은 고성능 프로세스뿐만 아니라 저전력 프로세서에서도 사용 가능하다. 마지막으로 FACE 기법의 경우 라운드 2 까지 사전 연산 테이블을 이용하여 단축이 가능하지만 제안하는 기법은 FACE 기법과 연동하여 사용하면 최대 라운드 3까지 연산을 최적화 할 수 있다.

[표 5-3] FACE와 제안 기법 비교

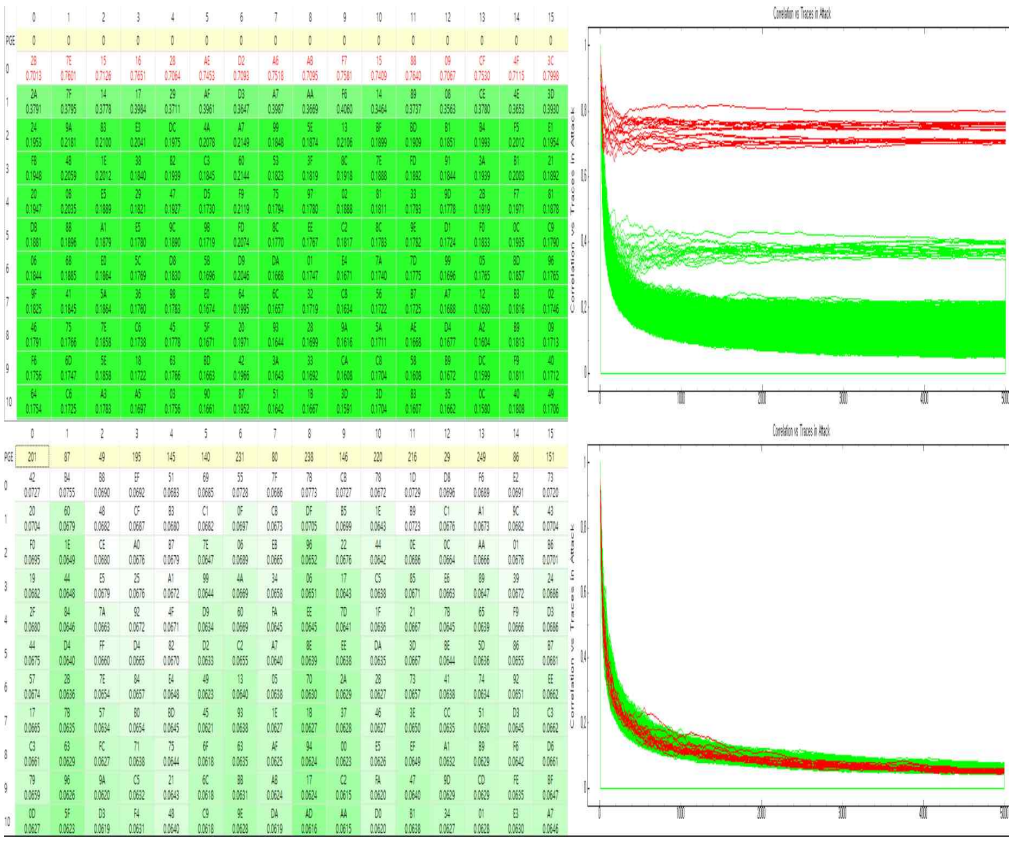
	FACE	**This Work**
Table Update	o	x
Constant Timing	x	o
Target Application	High-end processor	Low-end Processor and High-end processor
Supported Round	Round 2	Round 3

부채널 공격 기법이 알려진 이후로 소프트웨어로 암호를 구현할 때는 부채널 공격에 대응하기 위해 마스킹 연산을 추가하여 구현해야 한다. 작은 사이즈와 빠른 암호화 연산 시간을 가진 경량 암호 또한 부채널 공격 대비를 위해 마스킹 연산을 추기해야 한다. 하지만 대부분의 경량 암호들은 암호화 연산의 단순화를 위해 Add, Rotate, Xor 연산으로만 이루어진 ARX 구조로 이루어져 있다. 이러한 연산들은 마스킹 연산을 적용시키려면 산술(Arithmetic) 및 부울(Boolean) 연산에 의해 오버헤드가 상당히 커진다. 그에 반해 AES가 사용하는 SPN(Substitution Permutation Network) 구조의 경우 XOR 연산을 이용하여 단시간에 마스킹 연산 적용이 가능하다. 따라서 SPN 구조를 사용하는 AES가 다른 경량 암호에 비하여 마스킹 연산을 적용하는데 유리하다.

[표 5-4] LEA와 AES 마스킹 연산 성능 비교

Basic LEA-128	**Masked LEA-128**	**Masked AES-128 (This Work)**
168	2,286	388

[표 5-4]은 대표적인 ARX 구조의 경량 암호인 128-bit LEA의 일반 구현과 마스킹 연산을 적용한 구현 그리고 제안하는 기법에 마스킹 연산을 적용시킨 구현의 바이트 당 암호화 연산에 필요한 클록 사이클을 나타낸다. LEA 암호 알고리즘은 마스킹 연산을 추가하는 경우 상당히 큰 오버헤드가 발생하는 것을 알 수 있다. 그에 반해 AES 암호 알고리즘의 경우 오버헤드가 발생하긴 하지만 LEA와 비교했을 때 상당히 작은 것을 알 수 있다.



[그림 5-1] 일반 AES 암호에서 분석된 키 값 상관관계(上)
마스킹된 AES 암호에서 분석된 키 값 상관관계(下)

[그림 5-1]은 마스킹 연산이 적용되지 않은 일반 AES 암호와 마스킹 연산을 적용한 AES 암호에 CPA 공격을 통해 분석된 키 값과 그에 대한 상관계수를 그래프로 나타낸 것이다. [그림 5-1]의 상위 그래프는 마스킹 기법이 적용되지 않은 AES 암호로 실제 키 값이 매우 높은 상관 관계를 나타내어 공격자가 키 값을 쉽게 추측할 수 있다. 하지만 마스킹이 적용된 AES 암호의 경우 모든 예측 값의 상관 계수가 비슷한 것을 보아 공격자가 올바른 키 값을 추측할 수 없다.

5.2 128-bit 다항식 곱셈 최적화 기법 성능 평가

[표 5-5]에서는 제안하는 기법을 사용하여 최적화된 128-bit 이진 곱셈 연산을 이용한 AES-GCM 암호의 보안 단계별 성능 평가를 나타낸다. 128-bit의 보안 단계에서는 16-Byte, 64-Byte, 1024-Byte에 대해서 바이트당 클럭 사이클이 각각 807, 510, 415 클록 사이클인 것을 확인 할 수 있다. 이러한 특징은 192-bit, 256-bit의 보안 단계에서도 동일하게 적용된다. 제안 기법은 전체 암호화 과정 중 초기화 단계에서 사전 연산 테이블을 제작하는 과정을 거치기 때문에 짧은 메시지 암호화보다 긴 메시지 암호화에서 더 좋은 효율을 보인다.

추가적으로 속도 최적화가 아닌 최소한의 레지스터만을 이용해서 AES-GCM을 구현한 경우 속도 최적화 버전보다 성능이 2.8%~3.5%정도 느린 것을 볼 수 있지만 8-bit AVR 프로세서 및 다른 저전력 프로세서에도 적용시킬 수 있기 때문에 경쟁력을 가질 수 있다.

[표 5-5] AES-GCM 보안 단계별 성능 평가(* 레지스터 최적화)

Security Level	16-Byte	64-Byte	1024-Byte
128-bit	807	510	415
192-bit	868	548	446
256-bit	928	586	477
128-bit*	843	529	430
192-bit*	903	567	461
256-bit*	964	604	491

[표 5-6]에서는 128-bit 다항식 곱셈 연산 기법에 대한 이전 연구와 제안 기법의 성능 차이를 나타낸다. Seo와 Kim이 제안한 다항식 곱셈 기법의 경우 5,675 클럭 사이클이 필요한 반면, 제안하는 기법에서는 31.3% 줄어든 3,896 클럭 사이클이 필요하다.

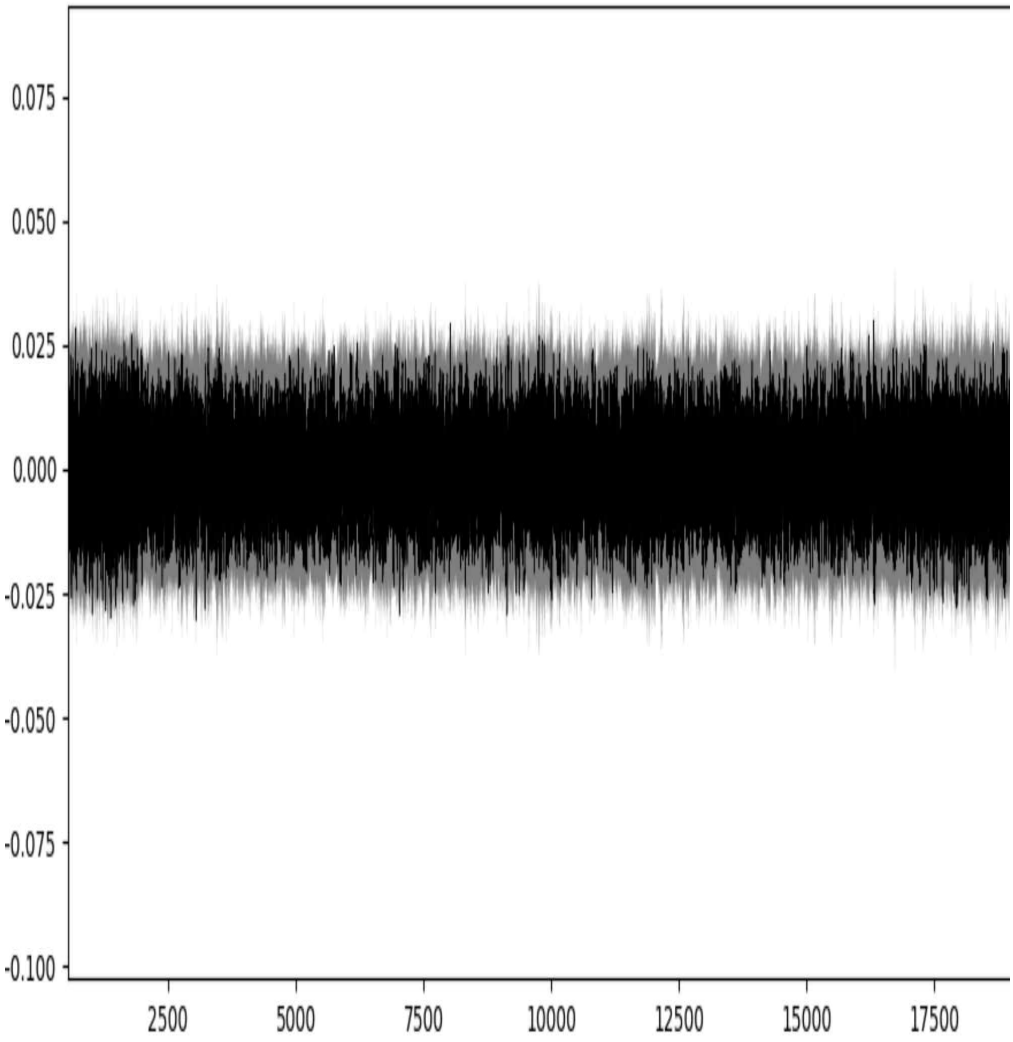
[표 5-6] 128-bit 다항식 곱셈 연산 성능 비교(* 레지스터 최적화)

Seo and Kim	This Work	This Work*
5,675	3,896	4,175

[표 5-7]에서는 128-bit 이진 곱셈 연산 기법에 대한 이전 연구들과 제안 기법의 성능 차이를 나타낸다. 제안하는 기법은 Seo와 Kim이 제안한 다항식 곱셈 기법에 비해 40.9% 줄어든 4,230 클럭 사이클이 필요하다. 또한 TA, SPA등의 부채널 공격에 대응성을 갖는다. 추가적으로 레지스터 사용을 최소화 하는 기법은 4,497 클럭 사이클이 소요된다.

[표 5-7] 128-bit 이진 곱셈 연산 성능 비교(* 레지스터 최적화)

Contributor	Method	TA/SPA Security	Timing
Liu et al.	Karatsuba + Masked Block-Comb	TA only	14,878
Seo and Kim	Karatsuba + Block-Comb with Dummy XOR and ILA	TA/SPA both	7,162
This Work	Karatsuba + Block-Comb with Dummy XOR and ILA	TA/SPA both	4,230
This Work*	Karatsuba + Block-Comb with Dummy XOR and ILA	TA/SPA both	4,497



[그림 5-2] GHASH에 대한 CPA 공격 그래프

[그림 5-2]은 GHASH 함수에 대해 Chipwisperer-Lite로 전력 소비량을 측정 후 측정된 파형을 기준으로 Python 3로 구현된 CPA 공격을 한 결과 값으로 예측되는 모든 키 값에 대한 상관 관계를 나타내는 그래프이다.

회색 파형은 실제 키 값이 아닌 다른 값들에 대한 상관 관계를 나타내고 검은색 파형은 실제 키 값에 대한 상관 관계를 나타낸다. 따라서 공격자는 실

제 키 값이 다른 값들에 비해 상관 관계 그래프가 두드러지지 않기 때문에 상관 관계를 보고 실제 키 값을 추측할 수 없다. 따라서 제안하는 기법은 CPA 공격을 성공적으로 방어할 수 있다.

Ⅴ. 결론

본 논문에서는 저전력 프로세서인 8-bit AVR 프로세서에서 AES-CTR 모드 및 128-bit 이진 곱셈 연산에 대한 최적화 구현 기법을 제안했다.

AES-CTR 모드의 최적화 구현은 사전 테이블을 이용한 최적화 구현 기법으로 AES 암호화 과정 중 라운드 2의 ShiftRows 연산까지 생략이 가능하다. 또한 이전 연구인 FACE 기법에 적용되어 최대 라운드 3의 ShiftRows 연산까지 생략 가능하여 성능을 향상한다.

그리고 AES-GCM에서 MAC 값을 생성하여 송신자를 인증하는 GHASH 함수를 최적화하여 구현했다. 부채널 공격에 대한 대응성을 갖추기 위해 사용되는 레지스터의 수를 최소화하고 확보한 레지스터를 Karatsuba 곱셈 알고리즘의 중간 결과 값들을 저장하는 버퍼 메모리로 사용하여 연산 시간을 크게 단축할 수 있었다. 또한 최소한의 레지스터만을 이용하여 128-bit 이진 곱셈 연산을 할 수 있는 레지스터 최적화 기법을 제안하여 레지스터 사용이 한정되는 다른 저전력 프로세스 상에서도 적용시킬 수 있다.

마지막으로 AES-GCM 연산 과정에서 AES-CTR에 대한 전력 소비량 분석 공격을 통한 부채널 공격을 방어하기 위해 암호화 연산에 마스킹 연산을 추가하여 구현하였고 GHASH에서 사용되는 128-bit 다항식 곱셈 과정에서 전력 소비량 분석 공격을 통해 해시 키 값이 유출되는 것을 방어하기 위해 마스킹 연산을 추가하여 부채널 공격에 대한 대응성을 갖추고 실제 공격을 방어함으로써 부채널 대응성을 증명했다.

향후 연구로는 저전력 프로세스 상에서 사용될 수 있는 다양한 경량 암호 및 운영 모드에 대한 최적화 구현 기법을 제시하고 부채널 공격에 대응할 수 있는 다양한 방어 기법에 대한 연구를 진행한다.

참 고 문 헌

1. 국내문헌

Park, E.; Oh, S.; Ha, J. Masking-Based Block Cipher LEA Resistant to Side Channel Attacks. J. Korea Inst.Inf. Secur. Cryptol. 2017, 27, 1023-1032.

2. 국외문헌

Hong, D.; Lee, J.; Kim, D.; Kwon, D.; Ryu, K.H.; Lee, D. LEA: A 128-bit block cipher for fast encryption oncommon processors. In Proceedings of the International Workshop on Information Security Applications,Jeju Island, Korea, 19-21 August 2013; pp. 3-27.

Hong, D.; Sung, J.; Hong, S.; Lim, J.; Lee, S.; Koo, B.; Lee, C.; Chang, D.; Lee, J.; Jeong, K. HIGHT: Anew block cipher suitable for low-resource device. In Proceedings of the International Workshop onCryptographic Hardware and Embedded Systems, Yokohama, Japan, 10-13 October 2006; pp. 46-59.

Beaulieu, R.; Treatman-Clark, S.; Shors, D.; Weeks, B.; Smith, J.; Wingers, L. The SIMON and SPECKlightweight block ciphers. In Proceedings of the 2015 52nd ACM/EDAC/IEEE Design AutomationConference (DAC), San Francisco, CA, USA, 8-12 June 2015; pp. 1-6.

Koo, B.; Roh, D.; Kim, H.; Jung, Y.; Lee, D.; Kwon, D. CHAM: a family of lightweight block ciphers forresource-constrained devices. In Proceedings of the International Conference on Information Security andCryptology, Seoul, Korea, 29 November-1 December 2017; pp. 3-25.

Roh, D.; Koo, B.; Jung, Y.; Jeong, I.W.; Lee, D.G.; Kwon, D.; Kim, W.H. Revised Version of Block CipherCHAM. In Proceedings of the International Conference on Information Security and Cryptology, Seoul,Korea, 4-6 December 2019; pp. 1-19.

Goubin, L. A sound method for switching between boolean and arithmetic masking. In Proceedings of theInternational Workshop on Cryptographic Hardware and Embedded Systems, Paris, France, 14-16 May2001; pp. 3-15.

Standard, N.F. Announcing the advanced encryption standard (AES). Fed. Inf. Process. Stand. Publ. 2001,197, 3-3.

McGrew, D.; Viega, J. The Galois/counter mode of operation (GCM). NIST Mod. Oper. Process 2004, 20,

Park, J.H.; Lee, D.H. FACE: Fast AES CTR mode Encryption Techniques based on the Reuse of RepetitiveData. IACR Trans. Cryptogr. Hardw. Embed. Syst. 2018, pp. 469-499.

Seo, S.C.; Kim, H. SCA-resistant GCM implementation on 8-Bit AVR microcontrollers. IEEE Access 2019,7, 103961-103978.

Kim, K.; Choi, S.; Kwon, H.; Liu, Z.; Seo, H. FACE-LIGHT: Fast AES-CTR Mode Encryption for Low-EndMicrocontrollers. In Proceedings of the International Conference on Information Security and Cryptology,Seoul, Korea, 4-6 December 2019; pp. 102-114.

López, J.; Dahab, R. High-speed software multiplication in f2m . In Proceedings of the InternationalConference on Cryptology, Calcutta, India, 10-13 December 2000; pp. 203-212.

Shirase, M.; Miyazaki, Y.; Takagi, T.; Han, D.G.; Choi, D. Efficient implementation of pairing-basedcryptography on a sensor node. IEICE Trans. Inf. Syst. 2009, 92, 909-917.

Seo, H.; Liu, Z.; Choi, J.; Kim, H. Karatsuba-Block-Comb technique for elliptic curve cryptography overbinary fields. Secur. Commun. Netw. 2015, 8, 3121-3130.

Liu, Z.; Seo, H.; Chen, C.N.; Nogami, Y.; Park, T.; Choi, J.; Kim, H. Secure GCM implementation on AVR.Discrete Appl. Math. 2018, 241, 58-66.

Gueron, S.; Kounavis, M. Efficient implementation of the Galois Counter Mode using a carry-less multiplierand a fast reduction algorithm. Inf. Process. Lett. 2010, 110, 549-553.

Dinu, D.; Biryukov, A.; Großschädl, J.; Khovratovich, D.; Le Corre, Y.; Perrin, L. FELICS-fair evaluation oflightweight cryptographic systems. In Proceedings of the NIST Workshop on Lightweight Cryptography,Gaithersburg, MD, USA, 4-6 November 2015; Volume 128.

Otte, D. AVR-crypto-lib. Available online: https://wiki.das-labor.org/w/AVR-Crypto-Lib/en (accessed on29 April 2020).

Seo, H.; Jeong, I.; Lee, J.; Kim, W. Compact implementations of ARX-based block ciphers on IoT processors.ACM Trans. Embed. Comput. Syst. (TECS) 2018, 17, 60.

ABSTRACT

Optimized implementation of AES-GCM on Low-End Microcontrollers

Kim, Kyung-Ho

Major in IT Convergence Engineering

Dept. of IT Convergence Engineering

The Graduate School

Hansung University

This paper proposes the techniques that can be implemented by optimizing Galois Counter Mode(GCM), a representative operating mode of Advanced Encryption Standard (AES), the most commonly used block cipher algorithm worldwide, on AVR 8-bit low-power processor. The AES-GCM is the most commonly used block cipher mode of operation for sending and receiving data with the ability to maintain confidentiality of the data and authentication to sender. In this paper, three optimization techniques are proposed to implement AES-GCM optimization. First, we replace the complex encryption process in AES-CTR mode, which is responsible for encryption in AES-GCM, with look-up table reference operations through pre-operating, improving performance by about 20% over previous methods. Second, we propose an optimization implementation technique using the minimization of register use and Karatsuba multiplication algorithm for the GHASH computation consisting of 128-bit binary multiplication in charge of authentication in AES-GCM. Lastly, We demonstrate the risistance to Side Channel Attack that is an attack technique that analyzes physical power consumption such as SPA(Simple Power Analysis), DPA(Differential Power Analysis)

and CPA(Correlation Power Analysis) attacks cryptographic algorithms by optimizing the masking operation to have risistance to SCA.

【주요어】 AES-GCM, Optimized implementation, Side Chnnal Attack, Karatsuba Algorithm, Low-end MicroController