Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.25, NO.4, Aug. 2015 ISSN 2288-2715(Online)
http://dx.doi.org/10.13089/JKIISC.2015.25.4.749

ARM-11 프로세서 상에서의 SHA-3 암호 알고리즘 구현 기술*

강 명 모,[1†] 이 희 웅,[2] 홍 도 원,[1‡] 서 창 호[1]
[1]공주대학교, [2](주)유엠로직스

Implementation of SHA-3 Algorithm Based On ARM-11 Processors*

Myeong-mo Kang,[1†] Hee-woong Lee,[2] Dowon Hong,[1‡] Changho Seo[1]
[1]Kongju National University, [2]UMLogics

요 약

스마트 시대가 도래함에 따라 스마트 기기의 사용이 점차 늘고 있다. 스마트 기기는 인류의 편의를 제공하여 널리 사용하고 있지만 정보가 노출될 위험이 존재한다. 이러한 문제를 보안하기 위해 스마트 기기는 자체적으로 다양한 암호 알고리즘이 포함되어 있다. 이 중 해시함수는 데이터 무결성, 인증, 서명 등의 알고리즘을 수행하기 위해 필수적으로 사용되는 암호 알고리즘이다. 최근 SHA-1의 충돌 저항성에 문제가 제기되면서 안전성에 문제가 생기게 되었고 SHA-1을 기반으로 한 현재 표준 해시함수인 SHA-2 또한 머지않아 안전성에 문제가 생길 것이다. 이에 따라 2012년 NIST는 KECCAK알고리즘을 새로운 해시함수 표준인 SHA-3로 선정하였고 이 알고리즘에 대한 다양한 환경에서의 구현이 필요해졌다. 본 논문에서는 SHA-3로 선정된 KECCAK 알고리즘과 기존의 해시 함수인 SHA-2를 ARM-11 프로세서에 구현하고 성능을 비교 분석하여 시사점을 도출하였다.

ABSTRACT

As the smart era, the use of smart devices is increasing. Smart devices are widely used to provide a human convenience, but there is a risk that information is exposed. The smart devices to prevent this problem includes the encryption algorithm. Among them, The hash function is an encryption algorithm that is used essentially to carry out the algorithm, such as data integrity, authentication, signature. As the issue raised in the collision resistance of SHA-1 has recently been causing a safety problem, and SHA-1 hash function based on the current standard of SHA-2 would also be a problem in the near future safety. Accordingly, NIST selected KECCAK algorithm as SHA-3, it has become necessary to implement this in various environments for this algorithm. In this paper, implementation of KECCAK algorithm. And SHA-2 On The ARM-11 processor, and compare performance.

Keyword: Hash function, SHA-2, SHA-3, ARM-11

I. 서 론

현대사회는 스마트화가 급속도로 진행되면서 스마트기기들을 주변에서 쉽게 접할 수 있게 되었다. 스마트기기들에는 필수적으로 암호 알고리즘이 내장되어 있는데 암호는 데이터를 주고받을 때 도청이나 데

접수일(2015년 5월 4일), 수정일(2015년 7월 8일), 게재확정일(2015년 7월 14일)
* 본 연구는 교육부와 한국연구재단의 지역혁신 창의인력 양성사업(No.2013H1138A2032077)과 2013년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대 정보·컴퓨팅기술 개발사업(No.2011-0029926)의 지원을 받아 수행된 연구임.
† 주저자, tkdaud1@kongju.ac.kr
‡ 교신저자,dwhong@kongju.ac.kr(Corresponding author)

이터를 가로채거나 중간자공격[1]에 의해 데이터의 원본이 수정될 수 있기 때문에 기본적으로 데이터의 기밀성과 무결성이 입증 되어야 한다. 이러한 요구사항을 만족시키기 위한 암호알고리즘들로 블록 암호, 공개키 암호, 해시함수, 난수 생성기 등이 존재하는데 이들 중 해시 함수는 메시지의 무결성이나 전자서명 등 다양한 곳에 사용되는 필수적인 알고리즘이다. 데이터를 주고받을 때 수신자와 발신자의 데이터의 해시 값을 서로 비교하면 주고받는 도중 데이터에 변경이 가해졌는지 원문 그대로인지를 확인하는 무결성 검증을 할 수 있기 때문이다. 이러한 암호학적 해시함수는 일방향성을 포함하고 있기 때문에 해시값을 이용하여 원문을 재현할 수는 없고, 역상저항성을 가지고 있어 같은 해시값을 가진 데이터를 만들어 내는 것도 어려워서 통신의 암호화 수단이나 인증, 디지털 서명에서 사용되고 있다. 해시 값이 만들어 지는 과정은 크게 2가지로 나눌 수 있다. 첫 번째는 전처리 과정이고 두 번째는 해시 계산 과정이다. 전처리 과정에서는 메시지 패딩 및 다음 단계에서 쓰일 값들을 초기화 하는 역할을 하고 있고 해시 계산 과정은 패딩 된 메시지로부터 메시지 스케줄링 과정을 여러 번 반복하여 메시지 다이제스트를 얻는 역할을 한다[2]. 하지만 이러한 해시 함수들 중 SHA-1이 2005년 중국의 Wang 교수에 의해 차분 공격을 이용한 충돌쌍 공격을 발견되면서 이와 유사한 구조를 가지고 있는 SHA-2 또한 안전성에 위협을 받을 우려가 생겼다[3]. 따라서 NIST(National Institute of Standards and Technology)에서는 2007년부터 공개경쟁을 통해 SHA-3 알고리즘을 공모하여 총 64개의 알고리즘이 후보에 등록되었고 2008년에 1차 후보로 51개의 알고리즘이 선정되었다. 2009년에 취약성이 드러난 후보들을 제외한 14개의 알고리즘이 2차 후보로 선정되었고 2010년 최종 후보로 5개의 알고리즘이 선정되어 2012년 최종적으로 KECCAK 알고리즘이 SHA-3 함수로 선정되었다[4]. 이렇게 선정된 SHA-3 알고리즘은 다양한 시스템에서 구현될 것으로 전망된다. 이미 스마트 카드인 UICC에서 SHA-3 알고리즘의 구현이 이루어진 상황에서[5] 스마트 기기와 비슷한 저 전력 디바이스에서도 구현이 이루어져야 할 것이다.

본 논문 2장에서는 SHA-3 알고리즘의 구조를 분석하고 3장에서는 임베디드 환경인 ARM에 대한 개요와 ARM-11 프로세서에서 SHA-3 알고리즘과 SHA-2 알고리즘을 구현하고 두 알고리즘의 성능을 비교분석한다. 마지막 4장은 구현 결과를 토대로 앞으로의 연구 방향에 대해 서술한다.

II. SHA-3 알고리즘

2.1 KECCAK 알고리즘의 선정

SHA-3 프로젝트는 SHA-1의 충돌쌍 공격이 발견되면서 안전성에 문제가 생기게 되어 현재 사용 중인 SHA-2 또한 머지않아 안전성에 문제가 생길 것에 대비하기 위해 기획되었다. 기존의 SHA-1과 SHA-2까지는 NIST에서 자체적으로 디자인했지만 SHA-3 알고리즘은 공개경쟁을 통해 후보를 모집하고 안전성을 분석하여 몇 차례에 걸쳐 후보를 줄여나가는 방식으로 진행되었다. 총 64개의 알고리즘이 등록되었고 Complete와 Proper 조건을 만족하는 51개의 알고리즘이 1차 후보에 선정되었다. Complete는 레퍼런스코드 및 최적화된 코드 구현 결과를 제출하는 것이고 Proper은 별도의 로열티 없이 전 세계적으로 누구나 사용 가능하고 다양한 하드웨어 및 소프트웨어에서 구현 할 수 있어야 함을 의미한다. 2차 후보에는 안전성, 알고리즘의 구현 효율성 그리고 알고리즘의 유연성과 단순성을 분석해 14개의 알고리즘이 선정되었다. 2010년 최종후보로 5개의 알고리즘이 발표되었고, 2012년 10월 1일 조앤 데먼, 질 반 아쉐, 마이클 피터스, 귀도 베르토니가 설계한 KECCAK이 SHA-3의 알고리즘으로 선정되었다.

2.2 KECCAK 알고리즘의 개요

KECCAK 알고리즘은 다이제스트의 길이가 정해져있는 SHA3-224, SHA3-256, SHA3-384, SHA3-512의 4개의 해시 함수와 다이제스트의 길이가 정해져있지 않은 SHAKE128, SHAKE256으로 불리는 XOF(Extandable Output Function)함수로 구성되어 있는데 XOF함수의 SHAKE에 붙는 숫자는 보안강도를 의미한다. 이 KECCAK 알고리즘은 스펀지 구조로 되어있기 때문에 스펀지 함수라고 불리며 흡수과정(absorbing)과 압착과정(squeezing)을 거쳐 해시값이 만들어진다. SHA-3 함수는 현재 SHA-2가 출력할 수 있는 다이제스트 크기를 모두 출력 할 수 있고 해시 함수가 갖추어야 할 역상 저항성, 제2 역상 저항성, 충돌 저항성을 모두 갖추고 있다. 또한 스펀지 구조와 이중구조의 유

연성을 가지고 효율적인 모드로 자생가능한 의사난수 비트 생성과 인증된 암호화에 사용가능하다. SHA-1, SHA-2와는 달리 KECCAK은 길이확장의 약점이 없기 때문에 HMAC 구간 설계가 필요하지 않고 MAC계산에서 키를 메시지 앞에 붙임으로 수행가능하다[6]. 게다가 라운드 함수에 대한 디자인 선택이 SHA-1과 SHA-2 그리고 블록암호의 대표적인 AES(Advanced Encryption Standard)와 상당히 차별성이 있다. 마지막으로 KECCAK은 하드웨어 성능이 우수하고 부채널 공격[7]에 대한 강점을 가지고 있다. 따라서 현재 SHA-2 알고리즘이 쓰이고 있는 모든 스마트기기에 SHA-3 알고리즘이 충분히 적용 가능 할 것이다.

2.3 SHA-3 알고리즘의 구조

2.3.1 KECCAK의 스펀지 구조

KECCAK 알고리즘은 그림 1과 같은 스펀지 구조로 되어있고 SPONGE$[f, pad, r](M, d)$로 나타낼 수 있다.

스펀지 구조는 b비트의 순열을 가지고 r비트의 크기를 입력으로 하는 f함수와 메시지를 입력비트인 r의 크기의 배수로 만드는 패딩함수로 구성되어있다. 그리고 평문인 메시지 M과 다이제스트의 길이인 d를 파라미터로 사용한다. f함수는 $f : \{0,1\}^b \rightarrow \{0,1\}^b$로 b비트의 State로 동작된다. b비트는 $\{25, 50, 100, 200, 400, 800, 1600\}$으로 정해져 있고 $b = r + c$로 나타낼 수 있으며 r은 f함수의 입력비트인 bitrate, c는 보안 파라미터를 의미한다. 메시지는 패딩 함수를 이용하여 입력비트인 r의 배수로 패딩되고 패딩한 메시지를 r의 크기만큼 나눈다. 각각의 r비트 메시지는 bitrate인 r과 XOR된 후 f함수에 입력된다. 이 과정이 반복되면서 각각의 r비트 메시지들이 absorbing되고 r비트 메시지가 모두 사용되면 압착과정(squeezing)으로 넘어간다. 압착과정(squeezing)에는 bitrate와 XOR연산은 하지 않고 f함수를 반복하여 원하는 다이제스트의 길이보다 크도록 이전 다이제스트에 연접하는 것을 반복한다[4].

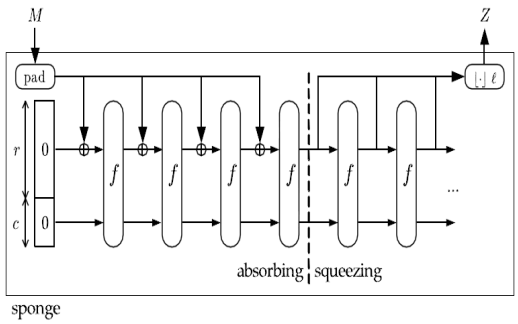

Fig. 1. The Sponge Structure

2.3.2 KECCAK 알고리즘의 State

KECCAK 알고리즘은 표 1과 같이 b의 값을 25로 나눈 w값과 w의 이진로그 값인 l로 구성된다.

Table 1. STATE widths and related w, l value

b(bit)	25	50	100	200	400	800	1600
w	1	2	4	8	16	32	64
l	0	1	2	3	4	5	6

w는 그림 2에서 볼 수 있듯이 f함수 내에서 치환되는 5×5×w의 3차원 행렬의 State를 만들어 낸다.

State는 기본적으로 $0 \le x < 5$, $0 \le y < 5$ 그리고 $0 \le z < w$의 값을 가지며 이 State의 형태로 f함수에 입력되어 메시지를 치환하는 과정을 거친다. 그림 3과 같이 x, y축 중간 값을 $(0, 0)$으로 잡고 z는 0부터 $w - 1$의 값을 가진다.

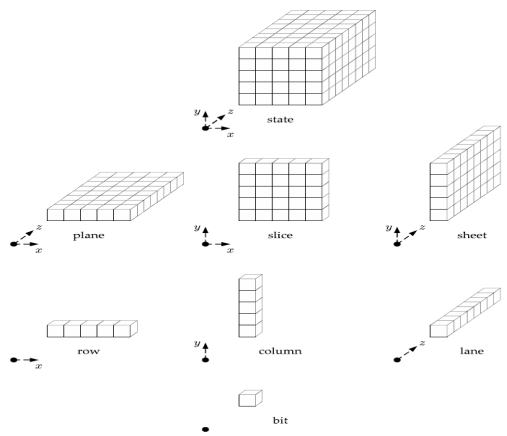

Fig. 2. Parts of the state array

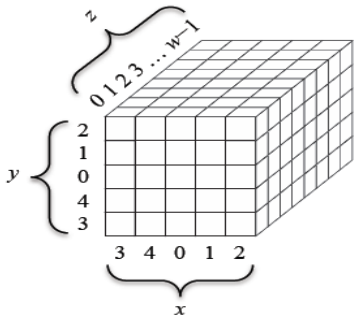

Fig. 3. Labeling Convention for the State Array

2.3.3 KECCAK 알고리즘의 f함수[8]

KECCAK 알고리즘의 f함수는 5단계의 Step mapping으로 구성되어있고 각 단계에서 A라는 State를 입력으로 하고 업데이트 된 A'을 출력하게 된다. Step mapping은 $\theta, \rho, \pi, \chi, \iota$의 알고리즘으로 구성되어 있으며 5단계를 거쳐 State 전체를 치환하게 된다.

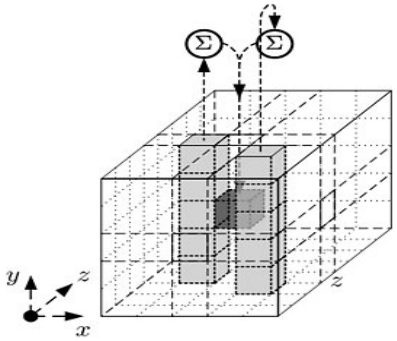

Fig. 4. Structure of $\theta(A)$

2.3.3.1 $\theta(A)$

그림 4에서 보듯이 하나의 칼럼을 XOR한 값 $A[x,0,z] \oplus A[x,1,z] \oplus A[x,2,z] \oplus A[x,3,z] \oplus A[x,4,z]$을 $C[x,z]$라 하고 모든 x, z에 대한 $C[x,z]$값을 구한다. 구한 $C[x,z]$값을 이용해 $C[(x-1) \bmod 5, z] \oplus C[(x+1) \bmod 5, (z--1) \bmod w] = D[x,z]$를 구하고 $A[x,y,z] \oplus D[x,z]$해줌으로 새로운 $A'[x,y,z]$로 치환해준다.

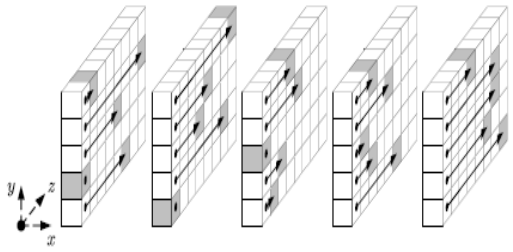

Fig. 5. Structure of $\rho(A)$

2.3.3.2 $\rho(A)$

그림 5와 같이 각각의 lane에서 offset 만큼 로테이션 되는 과정이다. 먼저 (x,y)를 $(1,0)$으로 잡고 $A'[x,y,z] = A[x,y,(z-(t-1)(t+2)/2) \bmod w]$를 계산해줌으로써 로테이션 된다. $(1,0)$에서 시작된 x, y값은 $(x,y) = (y, (2x+3y) \bmod 5)$의 식을 이용해 다음 로테이션 할 x, y를 구한다. t의 값은 0부터 시작해 23까지 x, y값이 바뀔 때 마다 1씩 증가시켜 x, y가 $(0,0)$을 제외한 24개의 레인을 로테이션 해준다. 그림 5의 offset은 b가 200일 때의 offset이다.

2.3.3.3 $\pi(A)$

그림 6과 같이 각 slice들의 부분들이 로테이션 되는 과정이다. $A[(x+3y) \bmod 5, x, z] = A'$의 식을 통해 로테이션 된다.

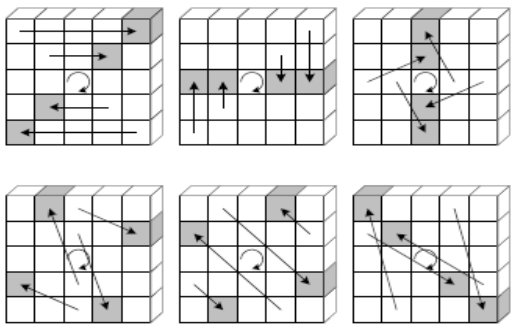

Fig. 6. Structure of $\pi(A)$

2.3.3.4 $\chi(A)$

그림 7과 같이 각각의 row들의 값을 치환하는 과정이다.$(x+1,y,z)$값과 $(x+2,y,z)$값을 AND 연산한 후 (x,y,z)값과 XOR한 값을 (x,y,z)에 저장한다.

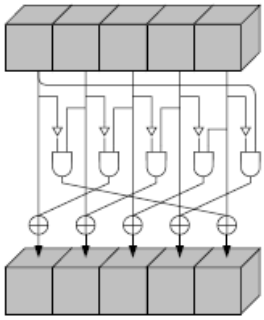

Fig. 7. Structure of $\chi(A)$

2.3.3.5 $rc(t)$

$\iota(A)$알고리즘에서 사용하는 라운드 상수를 만들어 내는 함수이다. 주어진 i_r(라운드 인덱스)를 이용하여 255개의 라운드 상수를 만들어낸다.

2.3.3.6 $\iota(A)$

$\rho(A)$알고리즘에서 로테이션 되지 않은 lane(0,0)을 로테이션 하는 과정이다. RC는 0으로 이루어진 w비트이고 $RC[2^j-1]=rc(j+7 i_r)$의 식을 통해 RC의 비트를 수정한다. j의 값은 0부터 l을 사용한다. 구한 RC값을 $A(0,0,z)$와 XOR 해줌으로 $A'(0,0,z)$로 로테이션 한다. ι의 효과는 i_r에 의존하는 방식으로 $lane(0,0)$의 비트들 중 일부를 변경하는 것이므로 다른 $24 lane$들은 ι함수의 영향을 받지 않는다.

위의 5가지 과정이 f함수의 한 라운드가 되고 $f(A,i_r)=\iota(\chi(\pi(\rho(\theta(A)))),i_r)$로 나타낸다. 여기서 i_r은 라운드 인덱스이다. 그리고 f함수는 b의 크기에 따라 $12+2l$의 라운드를 가지게 된다.

2.3.4 KECCAK 알고리즘의 패딩함수

메시지는 f함수 입력비트인 r비트의 배수가 되어야 하기 때문에 $\bmod r$ 연산을 이용한다. 메시지의 마지막 끝에 $j=(-m-2)\bmod r$ 로 구한 j 값을 이용해 $1||\ 0^j\ ||\ 1$을 연접하여 메시지를 패딩한다.

2.4 KECCAK 알고리즘의 안전성과 속도

KECCAK 알고리즘은 높은 수준의 병렬 구조로 되어있고 비교적 빠른 연산인 XOR과 AND를 f함수에 사용하고 있기 때문에 효율성이 좋다. 표 2를 보면 SHA-3가 MD5보다 좋은 효율성을 보여줄 뿐만 아니라 현재 사용하고 있는 SHA-2보다도 효율이 좋은 것을 확인 할 수 있다. 따라서 현재 우리나라에서 지정하여 사용하고 있는 해시함수 인 SHA-224, 256, 384, 512가 향후 SHA-3 알고리즘으로 대체 되어야 할 것으로 보인다.

Table 2. Performance of KECCAK and current hash algorithms[9]

Mib/s	Algorithm	Security
335	MD5	<64
192	SHA-1	<80
139	SHA-224	112
139	SHA-256	128
154	SHA-384	192
154	SHA-512	256
84	SHA3-256	128
66	SHA3-512	256

III. ARM-11프로세서에서 SHA-3 구현

3.1 ARM(Advanced RISC Machines)의 개요

ARM 아키텍처는 임베디드 기기에 많이 사용되는 32-bit RISC 프로세서이고 저 전력을 사용하도록 설계하여 ARM CPU는 모바일 시장에서 뚜렷한 강세를 보인다[10]. 본 논문에서 사용한 ARM프로세서는 ARM-11으로 ARMv6에 속한다. 이 프로세서는 현재 생산되는 대다수 스마트폰에 쓰이고 있으며 그 외에도 광범위한 컨슈머 제품, 가전제품, 임베디

드 영역에 쓰이고 있다. 또한 SIMD(Single Instruction Multiple Data) 소프트웨어 실행을 지원하는 미디어 명령이 포함되어 있으며 SIMD 명령들은 오디오 및 비디오 코덱을 포함하는 응용 프로그램들의 사용 확대를 위해 최적화되었다. ARM-11 프로세서는 소비 전력이 매우 낮고 ARM 보드의 크기에 따라 350MHz부터 시작하여 최적화하면 1GHz까지 속도를 낼 수 있다. 이러한 ARM-11 프로세서는 지금까지 나온 ARMv6 이하 프로세서와 호환이 가능하며 메모리 시스템, 예외 처리의 개선, 멀티프로세싱 환경을 위한 더 많은 지원을 한다.

특히 이번 연구에 사용한 프로세서는 Samsung의 32bit 어플리케이션 프로세서인 S3C6410가 내장된 보드로서 매우 넓은 범위에서 사용되고 있고 멀티미디어와 브라우저 기능, 보안 계산 환경을 제공하며 저렴한 가격에 1GBz의 성능을 낼 수 있다. 또한 보안을 위한 ARM TrustZone[11] 기술을 지원하며 높은 효율의 임베디드 자바 시스템을 위한 ARM Jazelle[12]를 지원한다. 게다가 메모리 시스템의 긴밀한 조합을 간소화하여 ARM-9프로세서의 이식과 실시간 설계가 가능하고 빠른 속도, 저전력에 알맞은 포터블 장치를 개발할 수 있게 한 개발 및 교육용 임베디드 실습 장비이다.

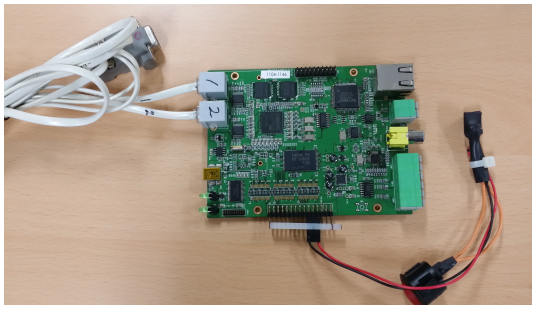

Fig. 8. ARM-11 processor

3.2 x86프로세서와 ARM 프로세서

일반적으로 스마트 기기는 저전력, 저렴한 비용, 저발열의 장점을 가진다. 특히 ARM 프로세서는 스마트폰이나 컨슈머제품같은 저전력, 저발열을 위한 좋은 장치이다. 반면 x86프로세서는 전력소모, 비용, 발열 면에서 ARM프로세서에 미치지 못한다. 이러한 차이가 나는 이유는 서로 CPU의 유형이 다르기 때문이다. ARM프로세서는 명령 집합과 구조가 단순하지만 x86프로세서는 복잡한 명령에 의해 동작되는 만큼 비싼 부품을 필요로 한다. 이처럼 타입이 다른 두 프로세서가 암호 모듈 성능에 얼마나 영향을 미치는지 확인해 봐야 할 것이다.

3.3 x86프로세서와 ARM프로세서에서의 구현

x86 프로세서 환경이 ARM 프로세서 환경보다 우수하지만, 본 논문에서는 각 프로세서 상에서 중점적으로 SHA-2와 SHA-3 알고리즘의 성능을 비교 분석 하였다. 표3은 x86과 ARM 프로세서의 사양이다. x86은 Intel Core i5-3470 CPU를 사용하고 최고 클록은 3.2GHz이다. 그리고 ARM 프로세서는 Samsung S3C6410 CPU를 사용하고 350MHz 클록의 코어를 내장하고 있다. ARM 프로세서는 32 bit 지원 형 프로세서이기 때문에 x86 32bit 운영체제 환경에서 연구를 진행하였고 컴파일러는 arm gcc version 4.3.5을 사용하였다. SHA-3의 state값인 b를 1600,800,400으로 지정하여 출력의 길이는 256bit와 512bit로 구현하였다.

또한 SHA2와의 성능을 비교하는 것이기 때문에 b가 1600일 경우 연산이 64bit워드로 진행되기 때문에 그림9와 같이 b가 800또는 400일 경우와 같은 32bit워드를 두 개 사용하여 구현하였고 따로 Squeeze 과정의 f함수는 구현하지 않았다. 패딩 함수는 출력이 256bit일 경우 메시지블록의 크기를 1088bit로 512bit의 경우 576bit의 크기로 f함수에 24번 반복되도록 한 뒤 마지막 메시지에만 메시지 블록의 크기가 되도록 패딩 하였다.

b가 800일 경우는 출력의 길이가 256bit일 때 메시지 블록은 544bit로 마지막 메시지 블록에 패딩을 해주고 f함수에 22번 반복되도록 하고 Squeeze 과정을 거친다. 출력의 길이가 512bit일 때도 마찬가지로 Squeeze 과정을 거치도록 하였다. 마지막으

Table 3. Specification of ARM-11 and x86

	ARM processor	x86 processor
CPU	Samsung S3C6410	Intel i5
Memory	1GB	4GB
Compiler	arm gcc ver 4.3.5	MS VS2010

Fig. 9. 1600 State of SHA-3 code and other states of SHA-3 code

로 b가 400일 경우 메시지 블록을 $144bit$로 하여 f함수에 20번 반복되도록 하고 Squeeze 과정을 이용해 $256bit$와 $512bit$를 출력하도록 하였다. SHA-2의 경우 출력의 길이가 $256bit$일 때는 연산을 $32bit$단위로 하기 때문에 메시지블록의 크기인 $512bit$에 맞게 패딩하고 64라운드를 반복하도록 하였다. $512bit$인 경우는 $256bit$의 출력과는 달리 연산이 $64bit$워드 단위로 되기 때문에 프로세서에 맞게 $32bit$워드연산으로 수정하고 메시지블록의 크기를 $1024bit$로 총 80번 반복되도록 출력 하였다.

그림 10은 Cross compiler[13]를 이용해 알고리즘을 컴파일 한 후 ARM-11 프로세서에서 실행시키는 화면이다. SHA-2 알고리즘과 SHA-3 알고리즘 비교방법은 메시지가 입력되는 시간을 t_0, 다이제스트가 출력되는 시간을 t_1으로 정하고 이것을 10,000번 반복했을 때 t_1-t_0인 t_d를 알고리즘의 성능으로 측정하였다. 일반적으로 암호 알고리즘의 성능 측정은 cycle-per-byte로 하지만 본 논문에서는 t_d를 구함으로써 ARM 프로세서 상에서 SHA-2 알고리즘과 SHA-3 알고리즘의 성능을 보다 정확하게 비교하였다.

표 4와 5에서 x86 프로세서와 ARM 프로세서에서의 SHA-2와 SHA-3 알고리즘의 성능을 확인 할

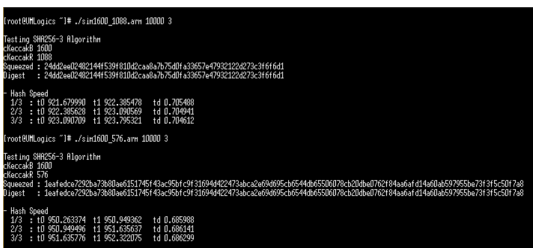

Fig. 10 Implementation screen on ARM-11

Table 4. Implementation of SHA-2 and SHA-3 on x86 (Unit : millisecond)

	SHA-3 (KECCAK)			SHA-2
	$b=1600$	$b=800$	$b=400$	
256 *bit*	0.0058	0.0035	0.0062	0.0127
512 *bit*	0.0056	0.0018	0.0118	0.0176

Table 5. Implementation of SHA-2 and SHA-3 on ARM-11 (Unit : millisecond)

	SHA-3 (KECCAK)			SHA-2
	$b=1600$	$b=800$	$b=400$	
256 *bit*	0.0704	0.0559	0.0894	0.3341
512 *bit*	0.0686	0.0289	0.1334	0.4672

수 있다. b가 400일 경우 r의 크기가 출력 길이인 $256bit$와 $512bit$보다 작기 때문에 만족하는 출력의 길이를 만들기 위해 f함수에 반복되므로 b가 1600인 경우와 800인 경우보다 속도가 느리게 나왔다.

그림 11과 12를 보면 출력 길이가 $256bit$인 경우 x86에서는 전체적으로 SHA-3 알고리즘이 SHA-2 알고리즘보다 2배에서 4배정도 좋은 성능을 보이고 ARM-11 프로세서에서는 SHA-3 알고리즘이 SHA-2 알고리즘보다 2.5배에서 최대 6배까지 성능이 좋은 것을 확인 할 수 있다.

출력 길이가 $512bit$인 경우 x86 프로세서는 SHA-3 알고리즘이 SHA-2 알고리즘보다 1.5배에

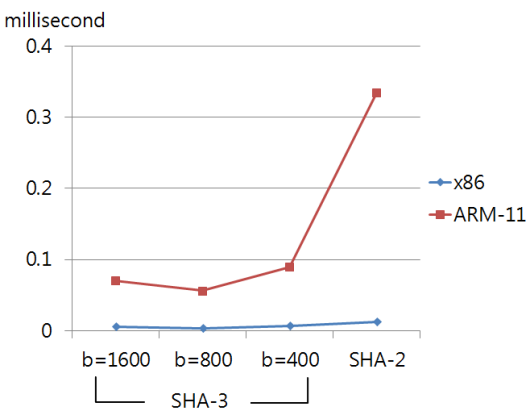

Fig. 11. Performance of Hash value $256bit$

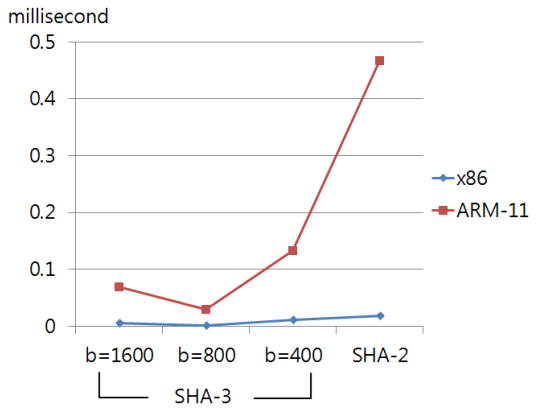

Fig. 12. Performance of Hash value 512bit

서 약10배 정도 좋은 성능을 보이고 ARM 프로세서에서는 3.5배에서 약16배의 좋은 성능을 보인다. 따라서 SHA-2와 비교한 SHA-3 알고리즘의 성능향상이 x86 프로세서보다 ARM-11 프로세서에서 훨씬 우수한 걸 확인 할 수 있다.

IV. 결 론

본 논문에서는 기존의 해시함수인 SHA-2와 SHA-3로 선정된 KECCAK 알고리즘을 저전력 임베디드인 ARM 프로세서와 x86 프로세서에서 각 프로세서의 성질에 맞게 구현하고 성능을 비교하였다. 그 결과 출력 길이가 256bit의 경우 x86 프로세서는 SHA-3 알고리즘이 최대 4배의 좋은 성능을 보였고 ARM-11 프로세서에서는 SHA-3 알고리즘이 최대 6배의 좋은 성능을 보였다. 또한 출력 길이가 512bit인 경우 x86 프로세서는 SHA-3 알고리즘이 최대10배, ARM-11 프로세서에서는 최대 16배의 좋은 성능을 보였다. 이것은 x86과 비교하였을 때 ARM-11 프로세서에서 SHA-3 알고리즘이 SHA-2보다 더 좋은 성능 향상 결과를 낸다고 할 수 있다. 이미 ARM 프로세서가 대다수 핸드폰이나 가전제품에서 쓰이는 만큼 기존의 SHA-2보다 안전성을 보장하고 성능면에서도 SHA-3 알고리즘을 사용하는 것이 바람직 하다. 이미 스마트 카드로 사용하고 있는 UICC에서도 SHA-2보다 SHA-3가 빠르다는 것을 알 수 있다[5]. 그러므로 SHA-3는 SHA-2를 대체하기에 충분한 알고리즘임을 확인 할 수 있고 앞으로 SHA-3알고리즘에 대한 이해와 각 환경에 맞도록 알고리즘을 최적화하는 연구가 진행되어야 한다.

References

[1] Y. Desmedt, "Man-in-the-middle attack," Encyclopedia of Cryptography and Security. Springer US, pp.759-759, 2011
[2] NIST, "FIPS PUB 180-4(SHS)" Mar. 2012
[3] X. Wang, A. C. Yao and F. Yao, "Cryptanalysis on SHA-1", CRYPTOGRAPHIC HASH WORKSHOP, Oct 31 - Nov 1. 2005
[4] NIST, "DRAFT FIPS PUB 202", May. 2014
[5] H Lee, D Hong, H Kim, C Seo & K Park, "An Implementation of SHA-3 Hash Function Validation Program and Hash Algorithm on UICC-16bit," Journal of KIISE, 41(11) pp.885-891, Nov. 2014
[6] MPG Bertoni, J Daemen, G Van Assche, "KECCAK (sha-3)" Icebreak2013, Jun. 2013
[7] R Novak, "Side-channel attack on substitution blocks," Applied Cryptography and Network Security. Springer Berlin Heidelberg, pp.307-318, Oct. 2003
[8] G.Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "The KECCAK reference Version 3.0," Citations in this document 14, Jan. 2011
[9] SHA-3,https://en.wikipedia.org/wiki/SHA-3
[10] O Yi, S Yun, M Park & H Song, "Implementation of ARIA Cryptographic Modules based on ARM9 Devices," International Journal of Security & Its Applications, 8(2) pp.243-250, Aug. 2014
[11] T Frenzel, A Lackorzynski, A Warg, H Härtig, "Arm trustzone as a virtualization technique in embedded systems," In Proceedings of Twelfth Real-Time Linux Workshop, Nairobi, Kenya. Oct. 2010
[12] M.A Fukase, P Khondkar & Nakamura. "Prototyping of a Java-embedded multimedia processor," Industrial Electronics

Society, 2001. IECON'01. The 27th Annual Conference of the IEEE. Vol. 3. IEEE, pp.2126-2130, Nov 29 - Dec 2. 2001

〔13〕 Cross_compiler, https://en.wikipedia.org/wiki/Cross_compiler

〈저자소개〉

강 명 모 (Myeong-Mo Kang) 학생회원
2015년 2월: 공주대학교 응용수학과 학사 졸업
2015년 3월~현재: 공주대학교 수학과 석사 재학
〈관심분야〉 암호이론 및 구현, 암호 알고리즘 구현

이 희 웅 (Hee-Woong Lee) 학생회원
2013년 2월: 공주대학교 응용수학과 학사 졸업
2013년 3월: 공주대학교 수학과 석사
2015년 3월~현재: (주)유엠로직스
〈관심분야〉 암호모듈 구현, 암호 알고리즘 구현

홍 도 원 (Dowon Hong) 종신회원
1994년 2월: 고려대학교 수학과 학사
2000년 2월: 고려대학교 수학과 박사
2000년 4월~2012년 2월: 한국전자통신연구원 팀장, 책임연구원
2012년 3월~현재: 공주대학교 응용수학과 교수
〈관심분야〉 암호기술, 프라이버시 보호기술

서 창 호 (Changho Seo) 종신회원
1990년: 고려대학교 수학과 학사
1992년: 고려대학교 수학과 이학석사
1996년: 고려대학교 수학과 이학박사
1996년~1996년: 국방과학연구소 선임연구원
1996년~2000년: 한국전자통신연구원 선임연구원, 팀장
2000년~현재: 공주대학교 응용수학과 교수
〈관심분야〉 암호알고리즘, PKI, 무선인터넷 보안 등