ISSN : 1226-7244 (Print)
ISSN : 2288-243X (Online)
논문번호 18-03-01
j.inst.Korean.electr.electron.eng.Vol.22,No.3,522~531,September 2018
http://dx.doi.org/10.7471/ikeee.2018.22.3.522

P-224 ECC와 2048-비트 RSA를 지원하는 공개키 암호 프로세서

A Public-key Cryptography Processor supporting P-224 ECC and 2048-bit RSA

성 병 윤*, 이 상 현*, 신 경 욱*★

Byung-Yoon Sung*, Sang-Hyun Lee*, Kyung-Wook Shin*★

Abstract

A public-key cryptography processor EC-RSA was designed, which integrates a 224-bit prime field elliptic curve cryptography (ECC) defined in the FIPS 186-2 as well as RSA with 2048-bit key length into a single hardware structure. A finite field arithmetic core used in both scalar multiplication for ECC and exponentiation for RSA was designed with 32-bit data-path. A lightweight implementation was achieved by an efficient hardware sharing of the finite field arithmetic core and internal memory for ECC and RSA operations. The EC-RSA processor was verified by FPGA implementation. It occupied 11,779 gate equivalents (GEs) and 14 kbit RAM synthesized with a 180-nm CMOS cell library and the estimated maximum clock frequency was 133 MHz. It takes 867,746 clock cycles for ECC scalar multiplication resulting in the estimated throughput of 34.3 kbps, and takes 26,149,013 clock cycles for RSA decryption resulting in the estimated throughput of 10.4 kbps.

요 약

FIPS 186-2에 정의된 224-비트 소수체 타원곡선 암호와 2048-비트 키길이의 RSA 암호를 단일 하드웨어로 통합 구현한 공개키 암호 프로세서 EC-RSA를 설계하였다. ECC의 스칼라 곱셈과 RSA의 멱승 연산에 공통으로 사용되는 유한체 연산장치를 32 비트 데이터 패스로 구현하였으며, 이들 연산장치와 내부 메모리를 ECC와 RSA 연산에서 효율적으로 공유함으로써 경량화된 하드웨어로 구현하였다. EC-RSA 프로세서를 FPGA에 구현하여 하드웨어 동작을 검증하였으며, 180-nm CMOS 셀 라이브러리로 합성한 결과 11,779 GEs와 14 kbit의 RAM으로 구현되었고, 최대 동작 주파수는 133 MHz로 평가되었다. ECC의 스칼라 곱셈 연산에 867,746 클록 사이클을 소요되어 34.3 kbps의 처리율을 가지며, RSA의 복호화 연산에 26,149,013 클록 사이클이 소요되어 10.4 kbps의 처리율을 갖는 것으로 평가되었다.

Key words : Public-key Cryptography, Elliptic Curve Cryptography, ECC, RSA, Information Security

---

* School of Electronic Engineering, Kumoh National Institute of Technology
★ Corresponding author
E-mail : kwshin@kumoh.ac.kr, Tel : +82-54-478-7427
※ Acknowledgment
・This work was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education(No. 2017R1D1A3B03031677)
・This research was supported by the KIAT(Korea Institute for Advancement of Technology) grant funded by the Korean government(MOTIE : Ministry of Trade, Industry and Energy). (No. N0001883, HRD Program for Intelligent Semiconductor Industry)
・Authors are thankful to IDEC for supporting EDA software.
Manuscript received Jul. 4, 2018; revised Sep. 5, 2018; accepted Sep. 11, 2018

Ⅰ. 서론

공개키 암호(public-key cryptography) 방식은 대칭키(symmetric-key) 암호, 해시(hash) 함수와 더불어 오늘날 유무선 통신 시스템의 보안 체계를 구성하는 핵심 요소이다. 공개키 암호는 정보 송신자와 수신자가 서로 다른 키(공개키와 비밀키)를 사용하므로 비대칭키(asymmetric-key) 암호 방식이라고도 하며, 사전에 비밀 키를 나눠 갖지 않아도 안전한 정보 송수신이 가능하다는 장점이 있다. 공개키 암호는 대칭키 암호 방식에 비해 1,000배 이상의 연산 복잡도를 가지므로, 대량의 정보를 암호화해야 하는 응용에는 적합하지 않으며, 전자서명 및 검증, 비밀키 교환 등에 사용된다. 향후, 스마트 카의 차량간 통신, 스마트 의료 및 헬스기기 보안, 사물인터넷(IoT) 등으로 응용분야가 확대될 것으로 예상된다[1-3].

대표적인 공개키 암호 방식으로 RSA(Rivest, Shamir, and Adleman) [4]와 타원곡선 암호(Elliptic Curve Cryptography; ECC) [5-6]이 있다. RSA는 매우 큰 소수의 곱으로 이루어진 정수의 인수분해가 어렵다는 사실에 안전성의 토대를 가지고 있으며 최초의 전자서명이 가능한 알고리듬으로 다양한 분야에 폭 넓게 사용되고 있다. ECC는 RSA 보다 짧은 길이의 키를 사용하면서도 비슷한 안정성이 얻어지므로, 많은 국제표준들이 기존의 RSA를 ECC로 바꾸는 추세이다. ECC는 International Organization for Standardization(ISO), American National Standards Institute(ANSI), National Institute of Standards and Technology(NIST), Standards for Efficient Cryptography Group(SECG) 등의 표준화 기구에서 공개키 암호 방식으로 채택하고 있으며, 또한 많은 상업용 제품에 탑재되고 있다.

최근 IoT, 무선 센서 네트워크(WSN), 모바일 헬스 및 의료기기 등과 같이 제한된 하드웨어 및 전력 자원을 갖는 응용분야의 보안에 적합하도록 경량 암호(lightweight cryptography) 방식에 관한 연구가 활발하게 이루어지고 있다. 경량 대칭키 암호와 경량 해시 함수에 관한 다양한 방식들이 제안되어 국제표준으로 채택되고 있으나, 경량 공개키 암호 방식에 관해서는 뚜렷한 성과가 없는 상태이다. 따라서 공개키 암호의 경량화 구현을 위해서는 ECC와 RSA의 저면적/저전력 하드웨어 구현이 필요하다.

본 논문에서는 대표적인 공개키 암호 방식인 ECC와 RSA를 동시에 지원하도록 단일 하드웨어 구조에 통합한 EC-RSA 프로세서를 설계하였다. ECC와 RSA에 공통으로 사용되는 핵심 연산장치를 32 비트 데이터 패스로 구현하였으며, 이들 연산장치와 내부 메모리를 ECC와 RSA 연산에서 효율적으로 공유함으로써 경량화된 공개키 암호 프로세서를 구현하였다.

2장에서는 공개키 암호 프로세서 설계를 위한 고려사항에 대해 간략히 설명하고, 3장에서는 ECC와 RSA를 단일 하드웨어로 통합하여 구현한 EC-RSA 프로세서의 구조 및 동작에 관해 설명한다. 설계된 회로의 FPGA 검증 결과와 성능평가를 4장에 기술하고, 5장에서 결론을 맺는다.

Ⅱ. 공개키 암호 프로세서 설계 고려사항

공개키 암호는 알고리듬적으로 서로 연관이 있는 한 쌍의 키(공개키와 개인키)를 사용하여 암호화·복호화하는 암호 방식을 말하며, ECC와 RSA가 대표적인 공개키 알고리듬이다.

RSA는 Ron Rivest, Adi Shamir, Leonard Adleman에 의해 개발된 공개키 암호 방식이며, 개발자 이름의 첫 글자를 따서 RSA로 명명되었다[4]. RSA는 큰 소수(prime number)의 곱으로 이루어진 정수의 소인수 분해가 어렵다는 사실에 보안성의 기반을 가지고 있으며, 정보의 암호·복호화뿐만 아니라 전자서명이 가능한 공개키 암호방식이다. 공개키(e, N)를 이용한 평문 M의 RSA 암호화는 식 (1)과 같이 모듈러(modular) 멱승으로 표현되며, 암호문 C가 얻어진다. 암호문 C를 개인키(d, N)로 RSA 복호화하면 평문 M이 얻어지며, RSA 복호화는 (2)와 같이 표현된다.

$$C = M^e \bmod N \tag{1}$$

$$M = C^d \bmod N \tag{2}$$

ECC는 Victor Miller와 Neil Koblitz에 의해 제안된 공개키 암호 방식이며 [5-6], 암호화·복호화는 식 (3)의 스칼라 곱셈(scalar multiplication)으로 구현된다.

$$Q = kP \tag{3}$$

ECC는 타원곡선 상의 임의의 두 점 P와 Q를 알더라도 비밀키로 사용된 임의의 정수 k를 알아내기 어렵다는 타원곡선 군(group)의 이산대수 문제에 안전성의 기반을 두고 있다.

식 (3)의 스칼라 곱셈을 계산하는 가장 기본적인 방법은 타원곡선 상의 점 P를 k번 더하는 것이다. 스칼라 곱셈의 연산량을 줄이기 위한 다양한 방법들이 제안되고 있으나 [7-8], 비밀키 k의 해밍 무게(Hamming weight)와 연산량이 상관관계를 가져 단순 전력분석과 같은 부채널 공격(side channel attack)에 취약하다는 단점이 있다. 본 논문에서는 스칼라 곱셈을 위해 몽고메리 래더(Montgomery ladder) 알고리듬 [9]을 사용하였다. 몽고메리 래더 알고리듬은 비밀키 k의 해밍 무게와 무관하게 동일한 횟수의 점 덧셈과 점 두배 연산을 반복하며, 연산량은 증가하지만 부채널 공격에 대한 내성을 갖는다.

타원곡선은 이진체(binary field) 또는 소수체(prime field) 상에서 정의될 수 있으며, 서로 장단점을 갖는다. 이진체 $GF(2^m)$ 상의 타원곡선은 상대적으로 하드웨어 구현에 유리한 장점을 갖지만, 소수체 $GF(p)$ 상의 타원곡선에 비해 상대적으로 큰 여인자(cofactor)를 가져 보안 성능이 다소 떨어진다. 소수체 상의 타원곡선은 작은 여인자를 가져 상대적으로 보안 성능이 우수하나, 하드웨어가 복잡해진다. 본 논문에서는 연산회로를 공유하여 RSA와 통합 구현하기 위해 소수체 상의 타원곡선을 지원하도록 설계하였다.

ECC의 스칼라 곱셈은 점 덧셈(point addition)과 점 두배(point doubling) 연산의 반복으로 계산되며, 점 연산은 타원곡선 상의 점을 표현하는 좌표계에 따라 하위의 유한체(finite field) 연산이 달라진다. 타원곡선 암호에는 아핀(affine) 좌표계 또는 투영(projective) 좌표계가 사용된다. 투영 좌표계는 3차원으로 변환하여 표현하는 방식으로 표준 좌표계, 자코비안(Jacobian) 좌표계, 로페즈-다합(Lopez-Dahab) 좌표계 등이 제안되고 있다[10-11]. 자코비안 좌표계를 이용한 점 연산은 모듈러 덧셈, 뺄셈, 곱셈, 제곱 연산으로 구현되며, 나눗셈 연산이 사용되지 않는다. 반면에, 아핀 좌표계의 점 연산은 모듈러 나눗셈 연산을 포함하므로, 하드웨어 복잡도가 높고 많은 클록 사이클을 필요로 한다. 본 논문에서는 자코비안 좌표계를 적용하여 나눗셈 회로 없이 저면적으로 구현하였다.

Table. 1. Comparison of ECC and RSA.
표 1. ECC와 RSA의 비교

	ECC	RSA
$GF(p)$ prime field	p: 224-b prime number	$N: p \times q$ $p \times q$: prime numbers
Operation	$Q = kP$	$C = M^e \bmod N$
Algorithm	Montgomery ladder	Left-to-right binary
Arithmetic component	Point addition : $C = A + B$ Point doubling : $C = 2A$	Multiplication : $C = A \times B$ Square : $C = A^2$
Finite field arithmetic	Multiplication, square, subtraction, addition	Multiplication, square
Operator	Word-based Montgomery multiplier, adder, subtracter	Word-based Montgomery multiplier

표 1은 ECC와 RSA의 알고리듬 특성을 비교한 것이며, 하위계층의 유한체 연산에 유사성이 있음을 알 수 있다. ECC는 점 연산을 기반으로 스칼라 곱셈이 수행되며, 점 연산은 소수체상의 덧셈, 뺄셈, 곱셈, 제곱 연산으로 구성된다. RSA는 소수체상의 곱셈과 제곱 연산으로 모듈러 멱승 연산이 수행된다. ECC와 RSA는 하위계층의 유한체 연산에서 모듈러 곱셈과 제곱 연산이 동일하다는 공통점을 갖는다. 따라서 다양한 피연산자(승수, 피승수) 길이에 대해 곱셈이 가능하도록 모듈러 곱셈기를 설계하면, ECC와 RSA를 통합하여 단일 하드웨어 구조로 구현할 수 있다. 본 논문에서는 ECC와 RSA의 핵심 연산장치인 모듈러 가산기와 곱셈기를 32 비트 워드기반으로 구현하였으며, 이를 통해 통합 공개키 암호 프로세서 EC-RSA를 경량 하드웨어로 구현하였다.

III. 통합 공개키 암호 프로세서 EC-RSA

1. 전체 구조

설계된 공개키 암호 프로세서 EC-RSA는 FIPS 186-2 [12]에 정의된 224-비트 소수체 타원곡선 P-224와 2048-비트 RSA를 지원한다. EC-RSA의 내부 구조는 그림 1과 같으며, 유한체 연산회로(GF_Alu), 두 개의 메모리 블록(ModP_Mem, RED_Mem) 그리고 제어블록으로 구성된다.

GF_Alu 블록은 ECC의 스칼라 곱셈과 RSA의 모듈러 멱승 연산에 필요한 모듈러 곱셈, 제곱, 가산/감산 연산을 수행하는 블록이며, 워드기반 몽고메리 곱셈기와 가산/감산기, 그리고 중간 결과값을 저장하는 내부 메모리 Alu_Mem로 구성된다. 메모리

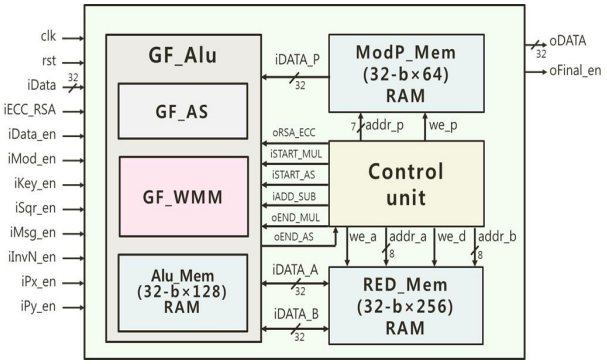


Fig. 1. Architecture of EC-RSA processor.
그림 1. EC-RSA 프로세서 구조

ModP_Mem은 유한체 연산에 사용되는 모듈러 값을 저장하며, 초기 모듈러 값이 저장되면 RSA와 ECC의 연산이 완료될 때까지 읽기동작만 수행된다. 메모리 RED_Mem은 RSA와 ECC 연산에 필요한 초기 벡터와 암호/복호화 연산의 중간 결과 값을 저장한다. 제어블록은 RSA와 ECC의 연산 알고리듬에 따라 GF_Alu 블록과 메모리의 동작을 제어하는 유한상태머신(finite state machine) 이다. 입/출력 포트와 내부 데이터 패스는 32 비트이며, ECC 또는 RSA에 따라 연산에 소요되는 클록 사이클 수가 달라진다.

2. 워드기반 모듈러 가산/감산기 (GF_AS)

GF_AS 블록은 32 비트 모듈러 가산과 감산 연산을 선택적으로 수행한다. 그림 2와 같이 두 개의 32 비트 캐리선택 가산-감산기(carry-select adder-subtracter) CSelAS와 캐리 저장 플립플롭, 모듈러 축약(modular reduction) 연산을 결정하는 로직 Det_Redt으로 구성되며, 신호 iAS에 따라 모듈러 가산 또는 감산 연산이 수행된다.

모듈러 축약 연산은 가산 또는 감산이 완료된 후에 처리하는 것이 일반적이나, 본 설계에서는 워드 단위의 가산/감산 연산 단계에서 모듈러 축약 연산이 함께 이루어지도록 구현하였다. 이와 같은 본 논문의 방법은 모듈러 축약 연산을 위한 비교기와 추가적인 클록 사이클이 필요 없다는 장점을 갖는다.

CSelAS_0는 32 비트 두 데이터 iDA와 iDB를 가산 또는 감산하여 32 비트 결과와 1 비트의 캐리를 출력한다. 캐리 출력은 플립플롭에 저장되어 다음 워드 연산의 입력과 Det_Redt 블록에서 축약 연산을 결정하기 위해 사용된다. CSelAS_1은 가산-감산기인

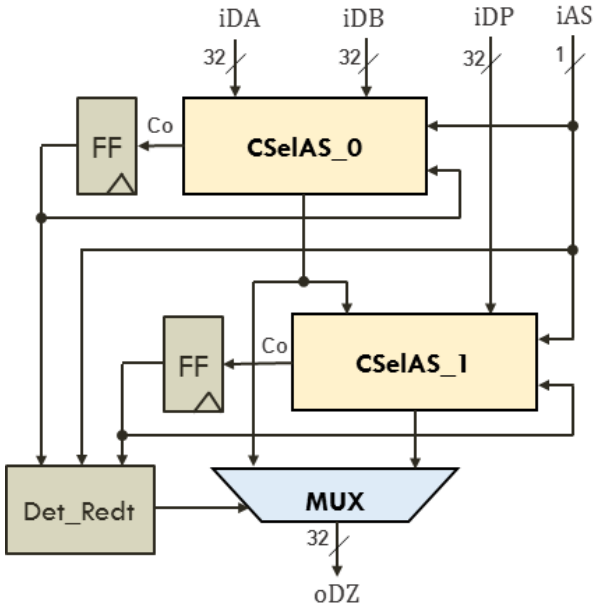


Fig. 2. GF_AS block.
그림 2. GF_AS 블록

CSelAS_0의 출력에 모듈러 값 iDP를 가산 또는 감산하여 모듈러 축약 연산을 수행한다. 모듈러 축약은 $(iDA+iDB)-iDP$ 또는 $(iDA-iDB)+iDP$ 의 연산을 통해 이루어진다.

3. 워드기반 몽고메리 곱셈기 (GF_WMM)

소수체 $GF(p)$ 상의 곱셈 연산은 interleaved 모듈러 곱셈 [13], 시프트-가산 곱셈 [7], 몽고메리 곱셈 [14] 등 다양한 알고리듬들이 제안되고 있다. 몽고메리 곱셈 알고리듬은 모듈러 축약을 효율적으로 수행하기 위한 방법이며, 최근에 몽고메리 곱셈 알고리듬을 하드웨어 구현에 적합하도록 최적화한 워드 기반 몽고메리 곱셈(Word-based Montgomery multiplication; WMM) [15, 16] 등이 연구되고 있다.

RSA와 ECC의 핵심 연산장치인 모듈러 곱셈기를 그림 3의 WMM 알고리듬을 기반으로 설계하였다. 워드기반 몽고메리 곱셈기는 유한체 곱셈을 고정된 크기(본 논문의 경우에는 32 비트)로 분할하여 반복 연산을 통해 유한체 모듈러 곱셈을 수행한다. 그림 3의 WMM 알고리듬은 문헌 [16]에 발표된 WMM 알고리듬에 비해 곱셈 연산에서 발생하는 중간 값의 워드 개수를 피연산자의 워드 개수와 동일해지도록 최적화하여 중간 값 저장 메모리를 줄였으며, 또한 내부 연산 루프가 $w+1$ 대신에 w 회에 완료되도록 최적화하여 소요 클록 사이클을

Input: $A = \{a_{w-1}, \cdots, a_1, a_0\}_{2^w}$
$B = \{b_{w-1}, \cdots, b_1, b_0\}_{2^w}$
$P = \{p_{w-1}, \cdots, p_1, p_0\}_{2^w}$
Pre_computed: $p_0^* = -p_0^{-1} \bmod 2^{32}$
Output: $S = (A \times B \times R^{-1}) \bmod P$

1: **for** $i = 0$ **to** $(w-1)$ **do**
2: $C1_{-1} \leftarrow 0, C2_{-1} \leftarrow 0$
3: $cy1 = cy2 = cy3 = cy4 \leftarrow 0$
4: $PP_0 \leftarrow a_i \times b_0$
5: $H \leftarrow s_0 + PP_0$
6: $q_i \leftarrow H \times p_0^* \bmod 2^{32}$
7: **for** $j = 0$ **to** $(w-1)$ **do**
8: $(C1_j, PP_1) \leftarrow a_i \times b_j$
9: $(cy1, H) \leftarrow PP_1 + C1_{j-1} + cy1$
10: **if** $j = w-1$ **then**
11: $(cy2, H) \leftarrow H + (s_{msb}, s_j) + cy2$
12: **else**
13: $(cy2, H) \leftarrow H + s_j + cy2$
14: **end if**
15: $(C2_j, PP_2) \leftarrow q_i \times p_j$
16: $(cy3, s_{j-1}) \leftarrow PP_2 + C2_{j-1} + cy3$
17: $(cy4, s_{j-1}) \leftarrow s_{j-1} + H + cy4$
18: **end for**
19: $(s_{msb}, s_{w-1}) \leftarrow cy1 + cy2 + cy3 + cy4 + C1_{w-1} + C2_{w-1}$
20: **end for**
21: **return** S

Fig. 3. Pseudo code of word-based Montgomery multiplication algorithm.
그림 3. 워드기반 몽고메리 곱셈 알고리듬의 슈도코드

줄였다. 그림 3의 슈도코드에서 w는 피연산자의 워드 개수를 나타내며, 타원곡선이 정의된 필드의 비트 크기 또는 RSA의 키 길이를 32 비트로 나눈 값이다. 예를 들어, 타원곡선 P-192의 경우에 $w = 192/32 = 6$이 된다.

그림 3의 슈도코드에서 $i-$루프는 승수 A의 워드 개수 w만큼 반복 연산을 구성하며, $j-$루프는 피승수 B의 워드 개수 w만큼 반복 연산을 구성한다. 승수 A의 특정 워드 a_i에 대해 $j-$루프에서 워드 단위로 부분곱 생성과 부분곱 가산이 이루어지며, 모듈러 축약 연산도 함께 이루어진다.

모듈로 값 P에 대한 축약 연산은 소수체 상의 합동(congruence) 특성을 이용한다. 이 때, 부분곱 가산 결과의 최하위 워드(least significant word : LSW)를 0으로 만드는 합동 값으로 변환하여 LSW를 제거하는 방법으로 축약 연산이 이루어진다. 이를 위해, 모듈로 값 P의 LSW p_0에 대한 곱의 역원의 음수 값 $p_0^* = -p_0^{-1} \bmod 2^{32}$이 사용되며, p_o^*를 이용하여 축약 연산을 위한 q_i 값이 계산된다. ECC

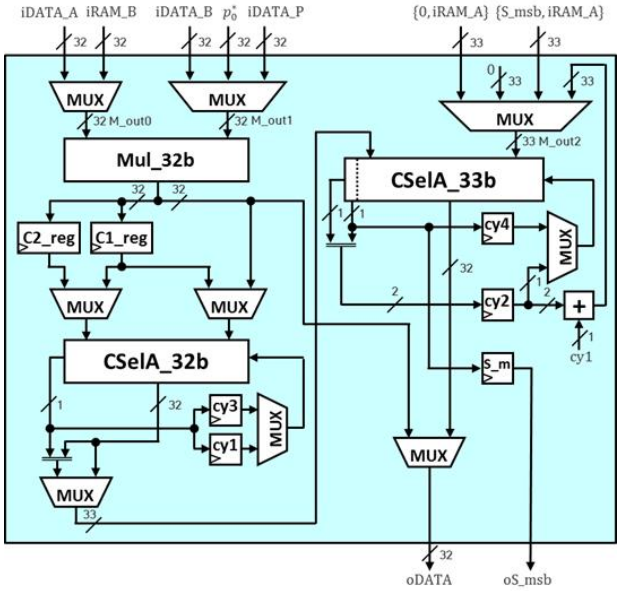


Fig. 4. Word-based Montgomery multiplier.
그림 4. 워드기반 몽고메리 곱셈기

의 경우에는 32'hFFFF_FFFF로 고정되며, RSA의 경우에는 외부에서 입력된다.

그림 3의 WMM 알고리듬 슈도코드에서 과정 4~6, 과정 8~14, 그리고 과정 15~17의 연산에 존재하는 규칙성과 유사성을 이용하면 효율적인 하드웨어 구현이 가능하다. 그림 3의 WMM 알고리듬을 적용하여 설계된 GF_WMM의 내부 구조는 그림 4와 같다. 32 비트 곱셈기 Mul_32b은 슈도코드의 과정 4, 8, 15의 곱셈연산을 워드 단위로 수행하여 64 비트의 부분곱을 만든다. 32 비트와 33 비트 가산기는 캐리 전파에 의한 지연을 최소화하기 위해 캐리선택 방식으로 구현하였다. 32 비트 캐리선택 가산기 CSelA-32b는 슈도코드의 과정 9, 16의 부분곱 가산 연산을 수행한다. 33 비트 캐리선택 가산기 CSelA-33b는 슈도코드의 과정 5, 11, 13, 17의 합동 값 계산을 위한 가산 연산을 수행한다.

그림 3의 WMM 알고리듬에 의한 곱셈 연산의 최종 결과는 $0 \le (S_{msb} S) < 2P$ 범위의 값을 가질 수 있다. 종래의 워드기반 몽고메리 곱셈기 [15, 16]은 최종 곱셈 결과가 모듈러 값 P 보다 큰 지를 판단하여 축약 연산이 추가적으로 수행되어야 한다. 본 설계에서는 워드기반 몽고메리 곱셈 결과가 GF_AS 블록으로 입력되어 축약 연산이 수행되며, 비교기와 뺄셈기가 사용되지 않아 추가 하드웨어 소모 및 클록 사이클을 줄였다.

입력 A, B에 대해, 일반적인 모듈러 곱셈의 결과

는 $C = A \times B \bmod P$이나, 몽고메리 곱셈의 결과는 $C = A \times B \times R^{-1} \bmod P$가 되며, 여기서 R은 $2^m \bmod P$이고, m은 타원곡선이 정의된 필드의 비트 크기 또는 RSA의 키길이를 나타낸다. 이와 같은 몽고메리 도메인 곱셈의 특성을 고려하여, 본 논문의 설계에서는 표 2와 같이 매핑 → 곱셈 → 역매핑의 3단계 과정으로 모듈러 곱셈이 계산되도록 설계하였다. 매핑 단계는 승수 A와 피승수 B에 각각 R^2을 몽고메리 도메인으로 곱셈하여 $AR \bmod P$, $BR \bmod P$로 변환하는 과정이다. 매핑된 AR와 BR을 몽고메리 도메인으로 곱셈하면 $ABR \bmod P$가 얻어진다. 마지막 역매핑 단계에서 ABR에 1을 몽고메리 도메인으로 곱하면 모듈러 곱셈 결과 $AB \bmod P$가 얻어진다.

4. 제어블록

설계된 EC-RSA 프로세서는 유한상태머신에 의해 전체적인 동작이 제어되며, 그림 5는 제어블록의 상태천이도를 보인 것이다. 초기 IDLE 상태에서 RSA 또는 ECC 모드신호와 함께 데이터 로드 신호에 의해 IN_DATA 상태로 천이되고, 각 모드의 데이터가 입력된다. 데이터 입력이 완료되면, MAPPING 상태로 천이되어 피연산자에 R^2을 몽고메리 도메인으로 곱셈하는 매핑과정이 수행된다. 매핑이 완료되면, ECC 또는 RSA 연산에 따라 RSA_MODE 또는 ECC_ MODE 상태로 천이되어 각 알고리듬에 따른 연산이 수행된다. 연산이 종료되면, OUT_DATA 상태로 천이되어 연산 결과가 외부로 출력된다.

ECC_MODE 상태에서는 그림 6-(a)의 FSM에 의해 동작이 제어된다. ECC의 키가 스캔된 후, K_ONE 상태와 K_ZERO 상태의 반복에 의해 스칼라 곱셈이 수행된 후, 투영 좌표계에서 아핀 좌표

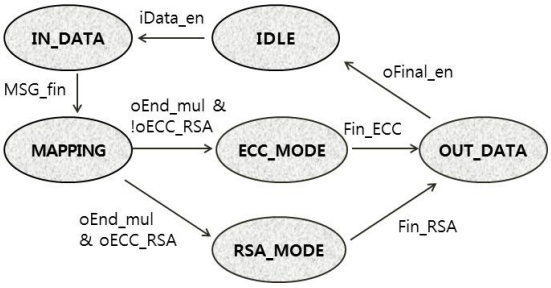


Fig. 5. Control FSM for EC-RSA Processor.
그림 5. EC-RSA 프로세서의 제어 FSM

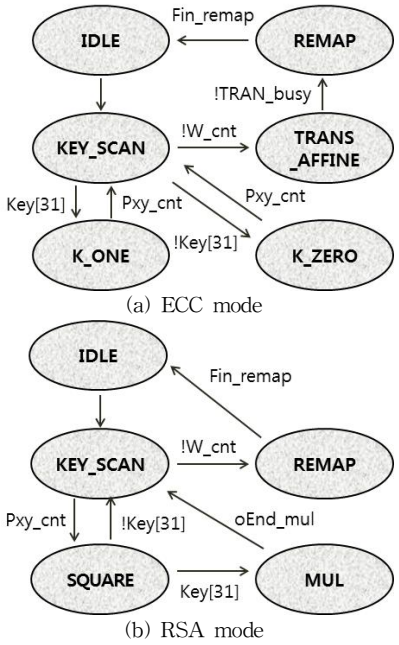


Fig. 6. Control FSM for ECC and RSA modes.
그림 6. ECC와 RSA 모드의 제어 FSM

계로의 좌표계 변환, 리매핑 순으로 연산된다. 스칼라 곱셈은 몽고메리 레더 알고리듬 [9]을 기반으로 구현되었다. 입력된 정수 k는 최상위 비트부터 스캔되며, 스캔된 k의 비트가 1이면 K_ONE 상태로, 0이면 K_ZERO 상태로 천이된다. K_ONE 상태와 K_ZERO 상태에서는 점 연산 수식에 따라 점 덧셈 연산 후, 점 두배 연산이 동일한 과정으로 수행된다. 정수 k의 모든 비트에 대한 처리가 완료되면 TRANS_AFFINE 상태로 천이되어 자코비안 좌표계 상의 점이 아핀 좌표계로 변환된다. 이과정은 스칼라 곱셈의 결과 값 x, y, z에 대해 x/z^2, y/z^3을 연산하여 아핀 좌표계 상의(x, y) 값으로 변환한다. z의 역원 연산은 페르마 소정리(Fermat's little theorem)를 적용하였다. 좌표계 변환이 완료되면 REMAP 상태로 천이되어 리매핑이 수행된 후, ECC_SCAN의 연산이 완료된다.

Table. 2. Montgomery modular multiplication process.
표 2. 몽고메리 곱셈 과정

Step	Operation
1. Mapping	$MMM(A, R^2) = A \times R^2 \times R^{-1} = AR \bmod P$
	$MMM(B, R^2) = B \times R^2 \times R^{-1} = BR \bmod P$
2. Multiplication	$MMM(AR, BR) = AR \times BR \times R^{-1} = ABR \bmod P$
3. Remapping	$MMM(ABR, 1) = ABR \times 1 \times R^{-1} = AB \bmod P$

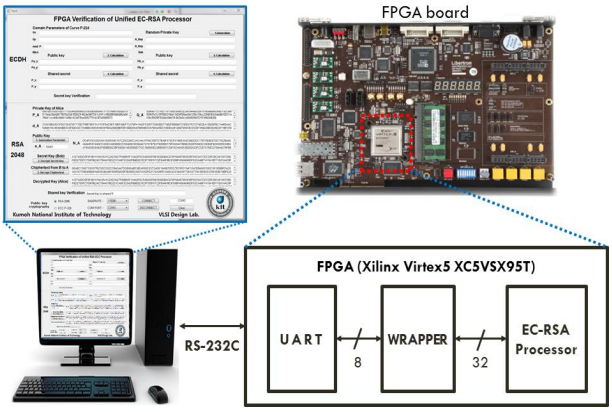


Fig. 7. FPGA verification setup for EC-RSA processor.
그림 7. EC-RSA 프로세서의 FPGA 검증 시스템

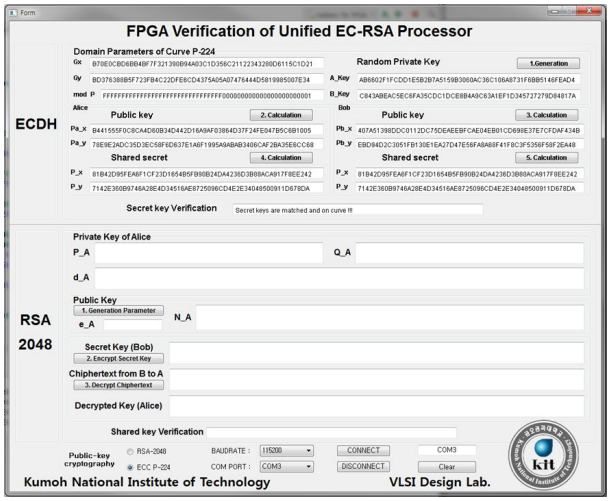
그림 6-(b)는 RSA_MODE의 동작을 제어하는 FSM이며, 입력된 메시지에 대해 식 (1), (2)의 멱승을 연산한다. 키 값의 최상위 비트부터 스캔하여 연산이 진행되는 left-to-right 이진 알고리듬을 적용하였다. 매 비트마다 제곱연산이 수행되고, 스캔된 키 값이 1이면 곱셈 연산이 추가적으로 수행된다. 키 값의 모든 비트에 대해 스캔이 완료되면 REMAP 상태로 천이되어 리맵핑이 수행된 후, RSA_MODE 연산이 완료된다.

Ⅳ. 기능검증 및 성능평가

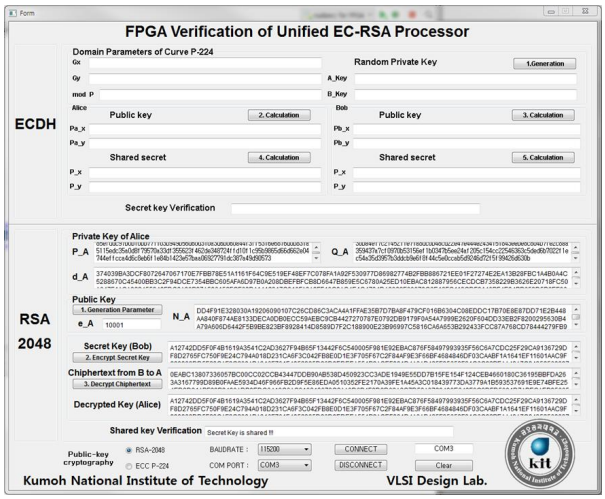
1. FPGA 구현을 통한 검증

Verilog HDL로 설계된 EC-RSA 프로세서는 FPGA 구현을 통해 하드웨어 동작을 검증하였다. FPGA 검증 시스템은 그림 7과 같으며, FPGA 보드, UART 인터페이스, Python 기반 구동 및 GUI 소프트웨어로 구성되며, FPGA 소자는 Xilinx Virtex5 XC5VSX-95T 디바이스가 사용되었다. FPGA에 구현된 EC-RSA 프로세서로 테스트 벡터를 로딩한 후, RSA 또는 ECC 연산을 수행한 결과가 PC로 전송되어 GUI 화면에 표시된다.

그림 8-(a)는 ECC 모드의 동작을 검증한 화면이며, ECDH(Elliptic Curve Diffie-Hellman) 키교환 프로토콜을 구현하여 설계된 EC-RSA 프로세서의 ECC 동작을 검증하였다. GUI 화면 상단 ECDH 영역의 버튼을 순서대로 클릭하면 다음의 과정으로 동작한다. 단계-① Alice와 Bob은 랜덤의 비밀키를 생성, 단계-②, ③ Alice와 Bob의 개인키 k_A, k_B와 생성점(Gx, Gy)가 FPGA 디바이스로 전송되



(a)



(b)

Fig. 8. Screenshots of FPGA verification results
(a) ECC mode (b) RSA mode.
그림 8. FPGA 검증결과 화면 (a) ECC mode (b) RSA mode

고 FPGA에 구현된 EC-RSA 프로세서에 의해 스칼라 곱셈이 수행되면, Alice와 Bob의 공개키(Pa_x, Pa_y)와 (Pb_x, Pb_y)가 생성되고 생성된 공개키는 상대방에게 전송된다. 단계-④, ⑤ Alice와 Bob의 개인키와 수신된 공개키가 다시 FPGA로 전송되어 스칼라 곱셈이 수행되면, 최종 결과인 Alice와 Bob의 공유키가 화면에 표시된다. ECDH에 의해 Alice와 Bob이 공유한 키 값은 소프트웨어 연산 결과와 비교되어 일치 여부가 화면에 표시된다. 이와 같은 검증을 통해 설계된 EC-RSA 프로세서의 ECC 모드가 올바로 동작함을 확인하였다.

그림 8-(b)는 EC-RSA 프로세서의 RSA 모드 동작을 검증한 화면이며, RSA를 이용한 키 전송 프로토콜을 구현하여 검증하였다. 최초 비밀키 생성

Table. 3. Comparison of public-key cryptography processors.
표 3. 공개키 암호 프로세서의 성능 비교

		This paper	[17]	[18]	[19]	[20]
Algorithm		ECC, RSA	ECC	ECC	RSA	RSA
RSA key size [bits]		2048	-	-	2048	2048
ECC field size [bits]		224	224	192	-	-
Throughput [kbps]		ECC: 34.3, RSA: 10.4	80.6	83.5	7.4	13.2
Technology / FPGA device		180 nm / Virtex-5	Virtex-4	Virtex-4	Virtex-6	180 nm
Hardware complexity	Gate Equivalence	11,780 GEs + 14 kbit RAM	-	-	-	12,540 GEs + 12 kbit RAM
	FPGA	588 slices + 3 BRAM	17,234 slices	8,590 slices	180 slices + 1 BRAM	-
Clock cycles for ECC scalar multiplication		867,746	147,285	113,472	-	-
RSA decryption time [ms]		196.6	-	-	277.26	154.9
Max. frequency [MHz]		133	53	48	447	165

을 위한 1024 비트의 소수 P와 Q를 입력한 후 단계-①에 의해 Alice의 공개키 e, 개인키 d, 모듈러 값 N, 그리고 Bob이 Alice에게 전송할 비밀키가 생성된다. 단계-② 생성된 파라미터와 Bob의 비밀키가 UART를 통해 FPGA 디바이스로 전송되고, FPGA에 구현된 EC-RSA 프로세서에 의해 Alice의 공개키를 이용한 모듈러 멱승 연산이 수행되면, 암호문이 화면에 표시된다. 단계-③ Alice에게 전송된 암호문과 Alice의 개인키가 FPGA로 전송되어 모듈러 멱승 연산을 통해 복호화가 수행되면, Bob이 전송한 비밀키가 복원된다. 이와 같은 검증을 통해 설계된 EC-RSA 프로세서의 RSA 모드가 올바로 동작함을 확인하였다.

2. 성능 평가

설계된 EC-RSA 프로세서를 180-nm CMOS 표준셀 라이브러리로 합성한 결과, 최대 동작 주파수는 133 MHz로 평가되었으며, 11,779 GE(gate equivalence)와 14 kbit의 RAM으로 구현되었다. 데이터 처리율은 ECC 모드에서 34.3 kbps, RSA 모드에서 10.4 kbps로 평가되었다.

문헌에 발표된 ECC, RSA 공개키 암호 프로세서 사례와 본 논문의 설계를 표 3에 비교하였다. ECC 또는 RSA를 구현한 사례들은 발표되고 있으나, ECC와 RSA를 단일 하드웨어로 통합 구현한 사례가 없어 간접적으로 비교한다. 문헌 [17], [18]의 사례는 각각 타원곡선 P-224, P-192를 지원하는 타원곡선 암호 프로세서이며, 80.6 kbps, 83.5 kbps의 처리율을 가져 본 논문의 ECC-RSA 통합 프로세서의 34.3 kbps 보다 높은 성능을 갖지만 더 큰 하드웨어를 필요로 한다. 문헌 [19], [20]의 사례는 2048-비트 키길이를 지원하는 RSA 프로세서 사례이다. 문헌 [19]의 경우는 본 논문의 설계에 비해 데이터 처리율이 낮고 복호화 소요시간이 더 길다. 문헌 [20]의 RSA 프로세서는 데이터 처리율, 면적, 최대동작 주파수 및 복호화 소요시간 등이 모두 본 논문의 설계와 비슷하나, 본 논문의 경우 ECC와 RSA를 동시에 지원하므로 유용성 측면에서 우수하다.

V. 결론

정보보안 시스템에 널리 사용되고 있는 대표적인 공개키 암호 알고리듬인 ECC와 RSA를 단일 하드웨어 구조로 통합하여 구현한 EC-RSA 프로세서를 설계하였다. ECC 모드에서 소수체 상의 224-비트 타원곡선 P-224와 RSA 모드에서 키길이 2048-비트를 지원한다. EC-RSA 프로세서를 180-nm CMOS 셀 라이브러리로 합성한 결과 11,779 GEs와 14 kbit의 RAM로 구현되었으며, ECC 모드에서 34.3 kbps, RSA 모드에서 10.4 kbps의 처리율을 갖는 것으로 평가되었다. ECC 또는 RSA 프로세서와 유사한 성능을 가지면서 두 가지 공개키 암호를 동시에 지원하므로, 유용성 측면에서 우수하다. 설계된 EC-RSA 프로세서는 제어회로의 FSM과 메모리를 추가하면 P-192, P-256, P-384 등 FIPS 186-2 표준에 정의된 타원곡선과 512/1024/3072-비트의

RSA를 구현하도록 확장될 수 있다. 핵심 연산장치와 메모리 공유를 통해 경량 하드웨어로 구현되어 제한된 자원을 갖는 IoT, 무선센서 네트워크(WSN) 등의 공개키암호 시스템 구현에 적합한 것으로 평가된다.

References

[1] H. Lin, and N. Bergmann, "IoT Privacy and Security Challenges for Smart Home Environments," *information,* pp.1-15, 2016. DOI:10.3390/info7030044

[2] O. Toshihiko, "Lightweight Cryptography Applicable to various IoT Devices," *NEC Technical Journal,* vol.12, no.1, pp.67-71, 2017.

[3] T. Eisenbarth and S. Kumar, "A Survey of Lightweight Cryptography Implementations," *IEEE Design & Test of Computers,* vol.24, pp.522-533, 2007. DOI:10.1109/MDT.2007.178

[4] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining Digital Signatures and Public-Key Crypto-systems," *Communications of the ACM,* vol.21, no.2, pp.120-126, 1978. DOI:10.1145/359340.359342

[5] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation,* vol.48, no.177, pp. 203-209, 1987. DOI:10.1090/S0025-5718-1987-0866109-5

[6] V. S. Miller, "Use of elliptic curve in cryptography," *in CRYPTO85: Proceedings of the Advances in Cryptology, Springer-Verlag,* pp.417-426, 1986.

[7] M. Amara and A. Siad, "Hardware implementation of Elliptic Curve Point Multiplication over GF(2^m) for ECC protocols," *International Journal for Information Security Research (IJISR),* vol.2, no.1, pp.106-112, March. 2012.

[8] F. Morain and J. Olivos, "Speeding up the computations on an elliptic curve using addition-subtraction chains," *RAIRO Theoretical Informatics and Applications,* vol.24, no.6, pp.531-543, 1990.

[9] P. L. Montgomery, "Speeding the Pollard and elliptic curve methods of factorization," *Mathematics of Computation,* vol.48, no.177, pp.243-264, 1987. DOI:10.1090/S0025-5718-1987-0866113-7

[10] J. H. Silverman, *The Arithmetic of Elliptic Curves,* Graduate Texts in Mathematics (GTM) 106, Springer-Verlag, 1986.

[11] J. López and R. Dahab, "Improved Algorithms for Elliptic Curve Arithmetic in GF(2^n)," *International Workshop on Selected Areas in Cryptography (SAC),* pp.201-212, 1998. Also in *Lecture Notes in Computer Science,* vol.1556, Springer.

[12] NIST Std. FIPS PUB 186-2, *Digital Signature Standard (DSS),* National Institute of Standard and Technology (NIST), Jan. 2000.

[13] J. Guajardo et al, "Efficient hardware implementation of finite fields with applications to cryptography," in *Acta Applicandae Mathematicae,* vol.93, pp.75-118, 2006. DOI:10.1007/s10440-006-9072-z

[14] Miyamoto et al, "Systematic design of high-radix Montgomery multipliers for RSA processors," *IEEE International Conference on Computer Design (ICCD),* pp.416-421, 2008. DOI:10.1109/ICCD.2008.4751894

[15] M. D. Shieh and W. C. Lin, "Word-Based Montgomery Modular Multiplication Algorithm for Low-Latency Scalable Architectures," *IEEE Transactions on Computers,* vol.59, no.8, pp.1145-1151, 2010. DOI:10.1109/TC.2010.72

[16] A. Bellemou, M. Anane, N. Benblidia, and M. Issad, "FPGA Implementation of Scalar Multiplication over F_p for Elliptic Curve Cryptosystem," *2015 10th International Design & Test Symposium (IDT),* pp.135-140, 2015. DOI:10.1109/IDT.2015.7396750

[17] K. Javeed, X. Wang, and M. Scott, "High performance hardware support for elliptic curve cryptography over general prime field," *Microprocessors and Microsystems,* pp.331-342, 2017. DOI:10.1016/j.micpro.2016.12.005

[18] K. Javeed, and X. Wang "FPGA Based High Speed SPA Resistant Elliptic Curve Scalar-Multiplier Architecture," *International Journal of Reconfigurable Computing,* pp.1-10, 2016. DOI:10.1155/2016/6371403

[19] B. Song, K. Kawakami, K. Nakano, and Y. Ito, "An RSA Encryption Hardware Algorithm

using a Single DSP Block and a Single Block RAM on the FPGA," *First International Conference on Networking and Computing,* pp.140-147, 2010. DOI:10.15803/ijnc.1.2_277

[20] W. L. Cho, and K. W. Shin, "2,048 bits RSA public-key cryptography processor based on 32-bit Montgomery modular multiplier," *Journal of the Korea Institute of Information and Communication Engineering,* Vol.21, No.8, pp.1471-1479, Aug. 2017.

BIOGRAPHY

Byung-Yoon Sung (Member)

2015 : BS degree in School of Electronic Engineering, Kumoh National Institute of Technology.
2017~ : Graduate student in School of Electronic Engineering, Kumoh National Institute of Technology.

Sang-Hyun Lee (Member)

2017 : BS degree in School of Electronic Engineering, Kumoh National Institute of Technology.
2017~ : Graduate student in School of Electronic Engineering, Kumoh National Institute of Technology.

Kyung-Wook Shin (Member)

1984 : BS degree in Electronic Engineering, Korea Aerospace University.
1986 : MS degree in Electronic Engineering, Yonsei University.
1990 : Ph. D. degree in Electronic Engineering, Yonsei University.
1990~1991 : Senior Researcher in Semiconductor Research Center, Electronics and Telecommunication Research Institute (ETRI).
1991~Present : Professor in School of Electronic Engineering, Kumoh National Institute of Technology.
1995~1996 : University of Illinois at Urbana-Champaign (Visiting Professor).
2003~2004 : University of California at San Diego (Visiting Professor).
2013~2014 : Georgia Institute of Technology (Visiting Professor).