



저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

석 사 학 위 논 문

224비트 ECDSA 하드웨어 기능 검증을 위한 소프트웨어 구현

지도교수 정 석 원

목포대학교 일반대학원

정보보호기술학협동과정

김 태 훈

2015년 2월

224비트 ECDSA 하드웨어 기능 검증을 위한 소프트웨어 구현

Software Implementation for 224bit ECDSA
Hardware Functional Verification

지도교수 정 석 원

목포대학교 일반대학원

정보보호기술학협동과정

김 태 훈

224비트 ECDSA 하드웨어 기능 검증을 위한 소프트웨어 구현

목포대학교 일반대학원
정보보호기술학협동과정

김 태 훈

상기자의 공학 석사학위 논문을 인준함.

소 속 직 위 성 명

심사 위원장 목포대학교 부교수

김 현 곤



심사 위 원 목포대학교 부교수

김 민 수



심사 위 원 목포대학교 부교수

정 석 원



2015년 2월

목 차

ABSTRACT

제1장 서 론	1
제2장 타원곡선을 이용한 전자서명	3
제1절 공개키 암호 시스템	3
1. 타원곡선 암호 시스템	3
2. 인수 분해의 문제를 기반으로 한 RSA	4
3. 이산대수 문제를 기반으로 한 ElGamal	5
4. 타원곡선암호와 타 암호 비교	5
제2절 타원곡선 암호시스템 구현을 위한 기초 지식	7
1. 소수 유한체	7
2. 타원곡선 스칼라 곱셈	9
3. ECDSA 지원 해시함수	11
제3절 타원곡선 전자서명(ECDSA)	14
1. 개요	14
2. ECDSA 서명 생성 및 서명 검증	14
제4절 ECDSA 하드웨어 구현 검증의 필요성	16
1. 암호 알고리즘 검증 방법	16
2. ECDSAVS	17
제3장 ECDSA 하드웨어 검증 소프트웨어	22

제1절 검증 소프트웨어 설계	22
1. 목표 ECDSA 파라미터	22
2. 목표 하드웨어 기본 구조	23
3. ECDSA 하드웨어 맞춤형 소프트웨어 설계 및 구현	25
제2절 ECDSA 하드웨어 내부 모듈 검증	28
1. 하드웨어 검증 방법	28
2. 유한체 사칙연산 모듈 검증	30
3. 스칼라 곱셈 모듈 검증	37
4. 타원곡선 점 연산 모듈 검증	39
5. SHA-1 모듈 검증	44
6. ECDSA 서명 모듈 검증	44
7. ECDSA 검증 모듈 검증	48
제4장 결 론	52
참 고 문 헌	53
국 문 초 록	56

표 목 차

<표 2-1> 공개키 암호 비교	5
<표 2-2> SHA-224의 메인 루프 연산과정	13
<표 3-1> 타원곡선 전자서명 파라미터 값	22
<표 3-2> 스칼라곱셈 알고리즘	25
<표 3-3> 유한체 덧셈과 뺄셈 테스트벡터 제공 함수	26
<표 3-4> 유한체 곱셈과 역원 및 모듈러 테스트벡터 제공 함수	26
<표 3-5> 타원곡선 위의 점 연산 테스트벡터 제공 함수	27
<표 3-6> ECDSA 전자서명 테스트벡터 제공 함수	27
<표 3-7> Adder 테스트벡터 예	30
<표 3-8> Sub 테스트벡터 예	31
<표 3-9> UmodP 테스트벡터 예	32
<표 3-10> CplusDmodP 테스트벡터 예	32
<표 3-11> UsubVmodP 테스트벡터 예	33
<표 3-12> Multiplier_modP 테스트벡터 예	34
<표 3-13> Invertor_modP 테스트벡터 예	35
<표 3-14> UmodN 테스트벡터 예	36
<표 3-15> dP 테스트벡터 예	37
<표 3-16> $2P$ 테스트벡터 예	39
<표 3-17> $P+Q$ 테스트벡터 예	41

<표 3-18> SHA224 테스트벡터 예	44
<표 3-19> ECDSA 서명 테스트벡터 예	45
<표 3-20> ECDSA 검증 테스트벡터 예	48

그 립 목 차

<그림 2-1> ECC, RSA, DSA 비교	6
<그림 2-2> 타원곡선 점 덧셈 ($P + Q = R$)	9
<그림 2-3> 타원곡선 점 곱셈 ($P + P = R$)	10
<그림 2-4> SHA-224 단계 연산 구조	13
<그림 2-5> 키 생성 테스트 과정	18
<그림 2-6> 공개키 검증 테스트 과정	19
<그림 2-7> 서명 생성 테스트 과정	20
<그림 2-8> 서명 검증 테스트 과정	21
<그림 3-1> k 비트 단위 덧셈	23
<그림 3-2> 224비트 덧셈기 하드웨어 구조	24
<그림 3-3> ECDSA 하드웨어 검증 소프트웨어 초기화면	27
<그림 3-4> ECDSA 내부 모듈	28
<그림 3-5> ECDSA 하드웨어 검증 순서	29
<그림 3-6> Adder 모듈 검증	31
<그림 3-7> Sub 모듈 검증	31
<그림 3-8> UmodP 모듈 검증	32
<그림 3-9> CplusDmodP 모듈 검증	33
<그림 3-10> UsubVmodP 모듈 검증	33
<그림 3-11> Multiplier_modP 모듈 검증	34
<그림 3-12> Multiplier_modP 하드웨어 시뮬레이션 결과	35
<그림 3-13> Invertor_modP 모듈 검증	35

<그림 3-14> UmodN 모듈 검증	36
<그림 3-15> dP 모듈 검증	37
<그림 3-16> dP 하드웨어 시뮬레이션 결과	38
<그림 3-17> dP 모듈 중간 값 검증	38
<그림 3-18> $2P$ 모듈 검증	39
<그림 3-19> $2P$ 모듈 중간 값 검증	41
<그림 3-20> $P+Q$ 모듈 검증	42
<그림 3-21> $P+Q$ 모듈 중간 값 검증	43
<그림 3-22> SHA224 모듈 검증	44
<그림 3-23> ECDSA 서명 모듈 검증	45
<그림 3-24> ECDSA 서명 중간 값	46
<그림 3-25> ECDSA 서명 하드웨어 시뮬레이션 결과	46
<그림 3-26> ECDSA 서명 다중 생성	47
<그림 3-27> ECDSA 서명 다중 생성 결과	47
<그림 3-28> ECDSA 서명 다중 생성 중간 값	47
<그림 3-29> ECDSA 검증모듈 검증	48
<그림 3-30> ECDSA 검증 중간 값	49
<그림 3-31> ECDSA 검증모듈 다중 검증	50
<그림 3-32> ECDSA 검증모듈 다중 검증 중간 값	50

Software Implementation for 224bit ECDSA Hardware Functional Verification

Tae Hoon Kim

*Interdisciplinary Program of Information & Protection, Mokpo
National University.*

(Supervised by Professor Seok Won Jung)

<Abstract>

Electronic signature verification method of elliptic curve in the CMVP verifies the input and output values for the entire module. If the verification fails, there is no feedback on what went wrong inside the module. Therefore, a major challenge is how to implement a hardware security module that varies as serial structure, parallel structure, pipeline structure, etc. For this, a software that provides different test vectors to be combined each structure respectively is required basically. More specifically, to satisfy these requirements, it is necessary to make an internal verification module required for the implementation of the Elliptic Curve Cryptography.

In this paper, we design novel software to verify the overall elliptic curve digital signature from the lower module to the upper module. The software provides different test vectors for entire internal module. One who makes cryptographic module can examine all internal modules in the phase of development process efficiently. Therefore, it can be expected to speed up in the development phase. In the near future, the software will be extended to provide more test vectors for other crypto modules except the ECDSA algorithm.

Keywords : ECDSA, CMVP, hardware verification, Verification Software

제1장 서론

타원곡선 암호의 효율성으로 인하여, 유한체 위에서의 타원 곡선을 하드웨어로 구현하기 위한 노력이 이루어지고 있다[8,10]. 타원곡선을 이용한 공개키 암호시스템은 공개된 키를 이용하여 인증, 전자서명 등의 서비스와 더불어 상호간의 공개된 키를 이용하여 비밀 값을 쉽게 유도할 수 있다는 장점이 있다[9]. 타원곡선 암호를 이용한 전자서명의 종류로는 ECDSA, EC-KCDSA 등이 존재한다. ECDSA는 타원곡선(Elliptic Curve)상에서 군(Group)을 정의하고 이에 대한 이산대수 문제의 어려움에 근거를 두고 있다[18]. 타원곡선 상에서의 이산대수 문제는 일반적인 군에서 정의되는 이산대수 문제보다 더욱 어렵다는 장점이 있으며, 작은 키로도 RSA 보다 높은 비도를 유지할 수 있기 때문에 많이 사용되고 있다[10].

타원곡선 암호 알고리즘에 대해 아핀 좌표계, 사영 좌표계를 이용한 다양한 방식[22]이 존재하며, 하드웨어 구조로는 직렬구조, 병렬구조, 파이프라인 구조 등 다양한 구조가 제안되고 있기 때문에 구현된 모듈이 제대로 동작하는지에 대한 검증이 매우 중요하다. 암호 모듈을 공식적으로 검증해주는 제도가 암호모듈 검증 프로그램인 CMVP (Cryptographic Module Validation Program)이다[20]. CMVP에서의 ECDSA 검증과정을 보면 키 생성 테스트, 공개 키 검증 테스트, 서명 생성 테스트, 서명 검증 테스트로 구분되어있다. 각 테스트는 모듈의 입·출력 값과 검증 모듈의 출력 값을 비교하여 이상 유무를 검증한다. 그러므로 모듈의 검증이 실패하였을 때 어떤 내부 모듈이 잘못되었는가에 대한 피드백은 없다. 또한, CMVP 이외의 연구는 PC와 주변기기의 병렬통신, 웹 어플리케이션 등 다양한 방식을 적용하여 상위 모듈을 검증하는 방법이 존재한다[15,16].

하드웨어 암호 모듈을 설계 후 검증까지 미치는 과정은 다음과 같다. 먼저 암호 모듈에 대한 하드웨어 구조를 설계한다. 설계된 부 모듈 별로 개발자가 테스트벡터를 만들어 기능을 검증하며 전체 모듈을 구현한다. 구현된 암호 모듈을 공인 검증을 받기 위해 CMVP 시험기관에 의뢰하면 시험기관

이 암호 모듈의 입·출력을 검증해 준다. 이 때 입·출력 시험에 실패했을 때 내부 모듈 어느 부분이 잘못되었는가에 대해서 찾기가 쉽지 않다. 따라서 하드웨어 설계 구조와 같은 구조로 소프트웨어를 구현하고 각 하드웨어 부 모듈의 개발 및 검증 단계에서 테스트벡터를 제공할 수 있는 소프트웨어 개발이 매우 필요하다.

본 논문에서는 ECDSA 검증 과정에서 서명 생성 및 서명 검증뿐만 아니라 개발 단계에서 내부모듈을 검증 할 수 있는 하드웨어 맞춤형 검증 프로그램을 구현하였다. ECDSA 하드웨어 모듈 검증용 소프트웨어는 키 길이가 224 비트인 ECDSA를 토대로 224비트의 유한체 사칙연산과 스칼라 곱셈, 타원곡선 전자서명까지의 내부 모듈 검증 기능을 개발하였다.

본 논문의 구성은 다음과 같다. 2장에서는 타원곡선 암호 알고리즘의 이해에 필요한 기본적인 사항을 서술하고 하드웨어 구현 검증의 필요성을 설명한다. 3장에서는 ECDSA 하드웨어 검증을 위한 목표 하드웨어의 구조와 이에 따른 하드웨어 맞춤형 소프트웨어에 대해 설명하고 4장에서는 결론을 다룬다.

제2장 타원곡선을 이용한 전자서명

제1절 공개키 암호 시스템

공개키 암호 방식의 RSA는 다양한 분야에서 응용되어왔으나 공개키와 비밀키를 생성하거나 복호화하기 위한 많은 계산을 필요로 한다. 타원곡선 알고리즘은 RSA 암호에 비해 강력한 암호 기술로 평가되고 있다. 이는 특정 알고리즘이 아닌 타원곡선이라는 수학 곡선을 활용한 기술로 RSA, Elgamal, Diffie-Hellman 등의 알고리즘을 기존의 정수 공간이 아닌 타원곡선 위에서 구현할 수 있게 한다[3].

이 장에서는 공개키 암호 시스템에서 대표적으로 사용되는 타원곡선 암호 알고리즘과 RSA, Elgamal의 암호 알고리즘을 비교해 보고 타원곡선을 이용한 암호 기술의 장단점에 대해 알아본다.

1. 타원곡선 암호 시스템

약 150년전부터 타원곡선(Elliptic Curves)은 수학적으로 광범위한 연구가 있어 왔다[17]. 타원곡선 암호 시스템은 비트당 안전도가 타 공개키 시스템보다 효율적이라는 것이 알려졌고, 최근 높은 속도의 구현이 가능하게 되었다.

유한체(finite fields)위에서 정의된 타원곡선 군에서의 이산대수 문제의 어려움을 기초로 한 타원곡선 암호시스템(ECC, Elliptic Curve Cryptosystem)은 1985년 N. Koblitz와 V. Miller에 의해 제안되었고, 현재 활발히 연구되고 있다. 타원곡선 암호시스템에서 1990년에 주목할 만한 성과 중의 하나가 Menezes, Okamoto와 Vanstone에 의해 연구되었다. 그들은 이 연구에서 초특이 타원곡선(Supersingular Elliptic Curve)의 이산대수 문제가 유한체 위에서의 이산대수 문제로 바뀔 수 있음을 보였다. 즉, Subexponential time algorithms이 존재한다는 것을 보였다. 그러므로 만약 완전지수복잡도(fully

exponential complexity)로 타원곡선 암호시스템이 깨지기를 원한다면, 초특이 타원곡선을 피해야 함을 의미한다. 그러나, 효율성 면에서 고려하면 초특이 타원곡선을 사용한 암호시스템이 몇 가지의 이점을 가지고 있으므로 무조건적인 배제는 옳지 않다.

타원곡선 암호는 유한체 상의 타원 곡선이 유한군을 가지며 그 위에서 이산대수 문제가 구성될 수 있음에 착안되어 제시된 암호 알고리즘이다. 이러한 타원 이산대수 문제를 기초로 다양하게 타원곡선을 활용하여 암호시스템을 설계할 수 있고, 암호시스템을 안전하게 설계하는 것이 용이하다[9]. 또한, 타원 이산대수 문제는 비교적 키의 크기를 작게 할 수 있는 실용상의 이점이 있다.

2. 인수 분해의 문제를 기반으로 한 RSA

정보화 사회를 구현하는 도구로 널리 사용되고 있는 RSA 암호화 기법은 인수분해 문제를 기반으로 설계되어 있고, 이에 따라 NP-Hard 문제인 정수의 소인수분해 알고리즘의 연구와 구현은 암호학에서 중요한 문제의 하나가 되어왔으며 지난 25년간 이 분야에서 많은 발전을 이룩하였다[14].

기존에 알려져 있는 인수분해 알고리즘으로는 ECM (Elliptic curve method), QS (Quadratic Sieve) 인수분해 알고리즘, MPQS (Multiple Polynomial Quadratic Sieve) 인수분해 알고리즘, 그리고 GNFS(General Number Field Sieve) 인수분해 알고리즘이 있는데 그 중에서 GNFS 인수분해 알고리즘이 큰 수를 대상으로 구현되는 경우 매우 효율적인 것으로 알려져 있다. 과거에 비해 효율적인 알고리즘이 개발되고 연산환경이 좋아짐에 따라 인수분해가 가능한 정수의 크기는 점점 커지고 있다. 그러나 자리수가 커지면 그 계산량이 많은 것이 단점으로 꼽힌다.

예를 들면, 비트 수에 따라 다르나 펜티엄급 컴퓨터에서 공개키와 개인키를 만들려면 짧게는 20여 초, 길게는 몇 분까지 기다려야 한다. 복호화에도 많은 계산량이 요구되고 있어 휴대용 단말기에서는 사용하기 어렵다. 또한, RSA가 정의되는 군의 대수적인 구조로 인해 구현하는 방법에 따라서 안전

성에 심각한 위협을 줄 가능성을 가지고 있다[3].

3. 이산대수 문제를 기반으로 한 ElGamal

T. ElGamal에 의해 소개된 ElGamal 암호 알고리즘은 RSA와 같은 공개 키 암호 알고리즘으로 다양한 분야에 활용되고 있다[13]. ElGamal 암호 알고리즘에서 사용되는 비밀키 길이는 다양한 길이로 설정할 수 있으며, 1024 비트 또는 2048비트 비밀키(pirvate key)가 많이 활용되고 있다. 이러한 ElGaml 암호 알고리즘은 이산 대수 문제(discrete logarithm problem)의 어려움에 기반하며 디피-헬먼(Diffie-Hellman) 키 동의 프로토콜과 유사하게 동작한다.

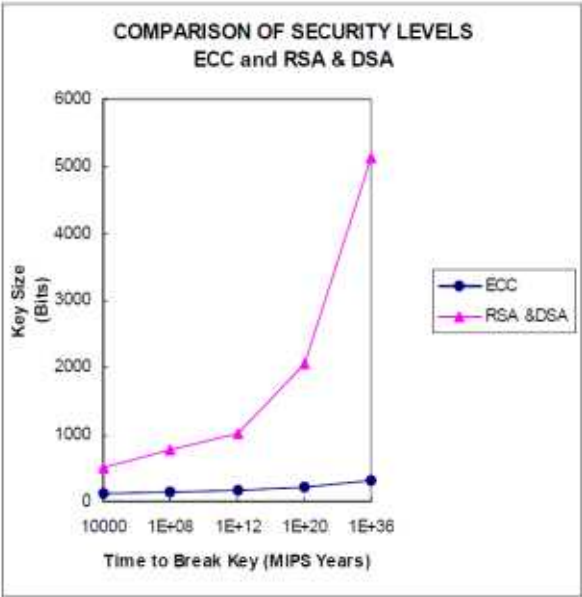
4. 타원곡선암호와 타 암호 비교

타원곡선 암호 시스템은 타원곡선의 이산대수 문제를 이용한 암호방식이며 RSA는 소인수분해의 문제, ElGamal은 이산대수 문제를 이용하였다. 세 가지의 암호 시스템은 다양한 기법을 통해 구현될 수 있다. 표준적인 방식으로 구현할 때를 예로 들어 타원곡선암호와 RSA, ElGamal를 비교하면 <표 2-1>과 같다.

<표 2-1> 공개키 암호 비교

	RSA	ElGamal	ECC
수학적 원리	소인수 분해	이산대수	타원곡선 이산대수
키 길이	1024~2048bit	1024~2048bit	160~256bit
속도	비교적 느림	비교적 느림	빠름
암호문 크기	-	평문의 두배	-
메모리	ElGamal에 비해 적음	많음	적음
비용	높다	높다	적다
통신	유선	유선	무선

<그림 2-1>에서 RSA와 ElGamal이 타원곡선암호에 비해 키 길이가 ECC에 비해 약 6배 가까이 길어 계산량이 많아 속도가 비교적 느리고, 메모리를 많이 차지하며, 비용이 높고 유선에 사용되는 것을 볼 수 있다.



<그림 2-1> ECC, RSA, DSA 비교

따라서 타원곡선 암호는 계산량, 키 크기 및 대역폭의 관점에서 타 공개 키 암호시스템보다 더 효율적이다. 구현에 있어 이러한 점은 높은 속도, 적은 전력 및 코드 크기를 줄일 수 있다.

제2절 타원곡선 암호시스템 구현을 위한 기초 지식

유한체 상에서 정의된 타원곡선에서의 이산대수 문제에 기초한 Elliptic Curve Cryptosystem은 타원곡선을 다양하게 활용함으로써 다양한 암호시스템을 설계할 수 있고, 암호시스템을 안전하게 설계하는 것이 용이하다[17]. 또한, 기존에 존재하는 공개키 암호화 알고리즘에 비해 짧은 키 길이를 가지고도 같은 안전도를 보장할 수 있다는 강점을 가지고 있다. 여기서는 Elliptic Curve를 암호화에 적용하는 방법에 대한 배경지식을 설명한다.

1. 소수 유한체

소수 유한체에 대한 내용을 참고문헌[22]에서 요약하면 다음과 같다. 유한체(finite field)는 원소의 개수가 유한한 체(field)를 뜻한다. 체(field)는 0으로 나누는 것을 제외한 사칙연산(덧셈, 뺄셈, 곱셈, 나눗셈)을 자유롭게 할 수 있는 집합을 말한다.

집합 $F (\neq \emptyset)$ 위에 덧셈 $+$ 과 곱셈 \cdot 이 정의 되어있고, 다음이 성립할 때 F 를 체라 한다.

- $(F, +)$ 는 아벨 군 이다.
- 곱셈 \cdot 에 대하여 결합법칙이 성립한다.
- 분배법칙이 성립한다.

만약 집합 F 가 유한한 경우, 유한체라고 한다.

p 를 소수라 할 때 소수체 $GF(p)$ 의 원소들 사이의 연산은 p 를 범으로 하는 모듈로 연산으로써 임의의 원소

$$\alpha, \beta \in GF(p)$$

에 대하여 덧셈은

$$\alpha + \beta = (\alpha + \beta) \bmod p$$

로 정의하며, 곱셈은

$$\alpha \cdot \beta = (\alpha \cdot \beta) \bmod p$$

로 정의한다. 뺄셈과 나눗셈은 각각 덧셈과 곱셈에 대한 역원을 구하여 더하거나 곱함으로써 구할 수 있으며, 역원은 확장 유클리드 알고리즘에 의해 효율적으로 계산될 수 있다.

NIST에서 발표한 FIPS 186-2에서는 5개의 소수체를 통해 타원곡선의 표준을 권장하고 있다[20].

$$\begin{aligned} p_{192} &= 2^{192} - 2^{64} - 1 \\ p_{224} &= 2^{224} - 2^{96} + 1 \\ p_{256} &= 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1 \\ p_{384} &= 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1 \\ p_{521} &= 2^{521} - 1 \end{aligned}$$

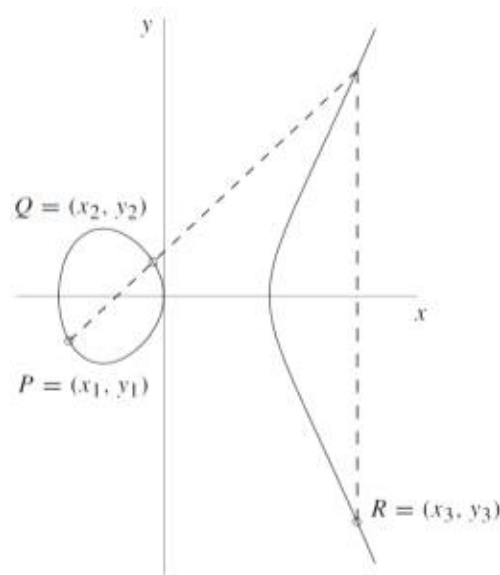
이러한 소수는 2의 멱수의 합 또는 차로 쓸 수 있는 특성을 갖는다. 또한, p_{521} 을 제외하고 지수승이 32의 배수이다. 이러한 속성은 워드크기가 32인 기계에서 특히 빠르기 때문에 절감된 알고리즘을 얻을 수 있다.

2. 타원곡선 스칼라 곱셈

필드 K 위에 정의한 타원곡선을 E 라고 하자. $E(K)$ 에 두 점을 연결하여 $E(K)$ 의 세 번째 점을 얻는 것이 chord-and-tangent rule이라 한다.

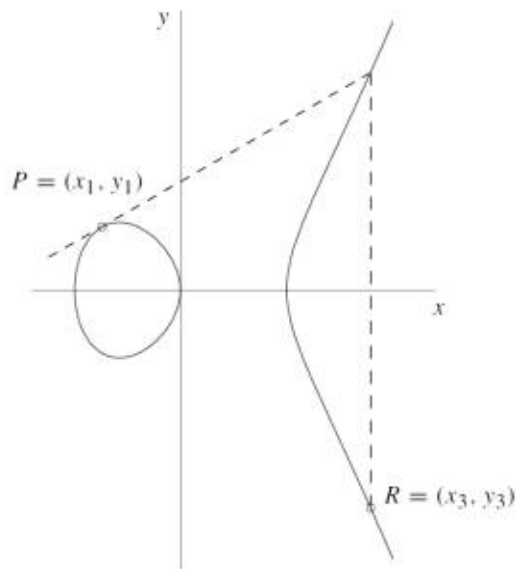
이 동작은 $E(K)$ 유형의 점들의 집합을 아벨 군 이라 한다. 이는 타원곡선 암호 시스템의 구성에 사용된다. 덧셈 방법은 가장 기하학적으로 설명하면, $P=(x_1, y_1)$ 이고 $Q=(x_2, y_2)$ 일 때 타원곡선 E 의 별개의 점에 존재한다. 이 점의 덧셈은 $R=P+Q$ 라고 정의된다.

먼저 P 와 Q 를 통해 선을 그린다. 이 라인은 세 번째 점에서 타원곡선을 교차하게 된다. 이어서 R 은 X 축을 중심으로 반사되는데 다음 <그림 2-2>와 같다.



<그림 2-2> 타원곡선 점 덧셈 ($P + Q = R$)

이어서 R 의 두 배 연산은 다음과 같이 정의된다. 먼저 P 에서 타원곡선의 접선을 그린다. 이 선은 두 번째 점에서 타원곡선이 교차한다. 이어서 R 은 x 축을 중심으로 반사되는데 다음 <그림 2-3>과 같다.



<그림 2-3> 타원곡선 점 곱셈 ($P + P = R$)

대수학적 공식의 군의 법칙은 기하학적 설명으로부터 도출될 수 있다. 바이어슈트라스 형태의 타원곡선 E 는 체 K 의 특성이 2 또는 3이 아닌 경우 (예를 들어, $K=\mathbb{F}_p$ 이고 $p > 3$ 인 소수)이다.

$E/K: y^2 = x^3 + ax + b$ 이며 $\text{char}(K) \neq 2, 3$ 인 형태의 군의 법칙은 다음과 같다.

- ① $P + \infty = \infty + P = P$ 이면 모든 P 는 $P \in E(K)$ 이다.
- ② 만약 $P = (x, y) \in E(K)$ 이면 $(x, y) + (x, -y) = \infty$ 이다. 점 $(x, -y)$ 는 $-P$ 로 표시되고 음수 P 라 부른다. $-P$ 는 타원곡선 $E(K)$ 의 한 점이고 $-\infty = \infty$ 이다.
- ③ 타원곡선 위 두 점의 덧셈과 한 점의 두배는 다음과 같다.

소수체 $\text{GF}(p)$ 위의 타원곡선 $E: y^2 = x^3 + ax + b$ 의 두 점 $P = (x_1, y_1)$ 과 $Q = (x_2, y_2)$ 에 대해

$$P+Q=(x_3,y_3), x_3=(\frac{y_2-y_1}{x_2-x_1})^2-x_1-x_2,$$

$$y_3=(\frac{y_2-y_1}{x_2-x_1})(x_1-x_3)-y_1$$

$$2P=(x_3,y_3), x_3=(\frac{3x_1^2+a}{2y_1})^2-2x_1,$$

$$y_3=(\frac{3x_1^2+a}{2y_1})^2(x_1-x_3)-y_1$$

으로 타원곡선 위의 두 점의 덧셈과 한 점의 두 배는 두 유한체 원소의 뺄셈, 한 유한체 원소의 역원, 두 유한체 원소의 곱셈, 한 유한체 원소의 제곱으로 구할 수 있다.

타원곡선의 스칼라 곱셈은

$$\begin{aligned} dP &= (d_{n-1}, d_{n-2}, \dots, d_0)P \\ &= (d_0P + d_12P + d_22(2P) + \dots) \end{aligned}$$

으로 2배 연산과 덧셈을 반복하면 구할 수 있다[22].

3. ECDSA 지원 해시함수

해시함수는 임의의 길이를 가진 입력 메시지를 고정된 길이의 출력으로 압축하는 함수로서 해시함수가 되기 위해서는 다음 조건을 만족해야 한다.

- 일 방향성(one-way) : 해시 값 y 에 대하여 $y=h(x)$ 를 만족하는 x 를 찾는 것은 계산적으로 불가능하다.

- 충돌 회피성(collision-free) : 주어진 x 에 대해 $h(x)=h(y)$ 를 만족하는 임의의 입력메시지 y 를 찾는 것은 계산적으로 불가능하다.

해시함수의 대표적 응용분야는 데이터의 무결성, 메시지 인증이다. 메시지에 대한 무결성을 확인하기 위해 송신자는 메시지와 함께 메시지의 해시 값을 함께 보내고 수신자는 전송 받은 메시지를 동일한 해시함수로 압축한 후 송신자로부터 받은 해시 값과 비교하여 메시지의 변형여부를 알 수 있다.

일반적으로 널리 쓰이는 해시함수 종류로는 MD5, SHA1, SHA-224, RMD160, Tiger, HAS160 등이 있고 SHA-383, SHA-512를 제외하고는 모두 입력 메시지를 512비트 블록단위로 처리하며, 마지막 블록의 패딩 규칙도 매우 유사하다.

SHA-224는 미국 NIST의 NSA에 의해서 개발된 MD5와 유사한 구조로 설계되었으나 보다 안전한 것으로 인정받고 있다. MD5와는 달리 224비트의 출력을 가지고, 512비트 입력에 대해 64번의 단계 연산을 행한다. 64개의 메시지 워드를 생성함에 있어 처음 15개까지는 512비트 입력 메시지를 그대로 입력하고 그 다음부터는 아래와 같은 연산을 통하여 얻은 값으로 입력한다.

$$W^t = M^t \quad (0 \leq t \leq 15)$$

$$W^t = \sigma_1(W^{t-2}) + (W^{t-7}) + \sigma_0(W^{t-15}) + (W^{t-16}) \quad (16 \leq t \leq 63)$$

이러한 연산의 메인 루프의 연산과정은 다음과 같다.

<표 2-2> SHA-224의 메인 루프 연산과정

$$a = H^0, b = H^1, \dots, h = H^7$$

For $t = 0$ to 63

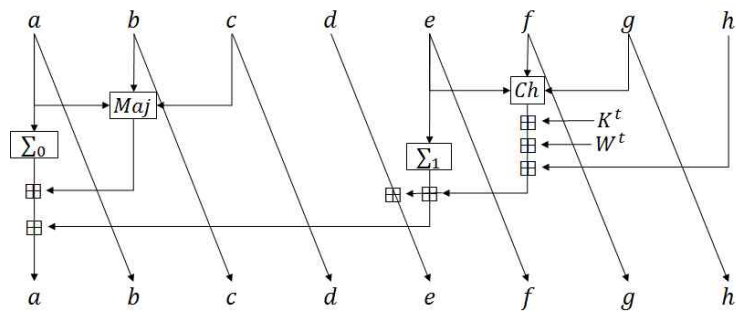
$$T_1 = h + \Sigma_1(e) + Ch(e, f, g) + K^t + W^t$$

$$T_2 = \Sigma_0(a) + Maj(a, b, c)$$

$$h = g, g = f, f = e, e = d + T_1, d = c, c = b, b = a, a = T_1 + T_2$$

$$H^0 = a + H^0, H^1 = b + H^1, \dots, H^7 = h + H^7$$

따라서 앞서 설명한 연산방법을 토대로 SHA-224의 연산 구조는 다음과 같다.



<그림 2-4> SHA-224 단계 연산 구조

제3절 타원곡선 전자서명(ECDSA)

1. 개요

타원곡선 암호의 장점들로 인해 스마트 카드나 무선통신 단말기 등과 같이 메모리와 처리능력이 제한된 응용분야에서 특히 효율적으로 사용될 수 있다. 이에 따라 각종 국제 표준들에서 타원곡선 암호에 대한 표준화가 활발히 진행되고 있으며, 또한 실제로 다양한 보안응용들에서도 타원곡선암호를 지원하고 있다.

현재 가장 널리 사용되는 타원곡선 암호시스템은 크게 전자서명과 키 교환으로 나눌 수 있다. 전자서명은 미국 연방 표준인 DSA(Digital Signature Algorithm)를 타원곡선 위로 적용한 ECDSA(Elliptic Curve DSA)가 국제적으로 가장 널리 사용되고 있으며, 국내에서도 기존의 전자서명 표준 KCDSA(Korean Certificate - based Digital Signature Algorithm)의 타원곡선 변형을 EC-KCDSA(Elliptic Curve KCDSA)라는 이름으로 TTA 표준으로 제정한 바 있다. 키 교환 알고리즘으로는 ANSI X9.63의 ECDH (Elliptic Curve Diffie-Hellman)알고리즘이 가장 널리 사용된다. 이들 알고리즘은 여러 표준에 포함되어 있는데 표준의 범위에 따라서 약간씩 차이를 보이고 있으므로 상호 호환성을 위해 표준의 성격을 정확히 이해하고 구현하는 것이 필요하다.

2. ECDSA 서명 생성 및 서명 검증

서명 과정을 살펴보자. 타원곡선 전자서명은 n 을 타원곡선 군의 위수라고 할 때 메시지 m 에 대한 서명과 검증이 다음과 같이 이루어진다. 서명자의 공개키 Q 는 비밀 키 d 로부터 $Q = dG$ 로 계산된다[3].

서명 생성과정은 다음과 같다[27].

- ① $\{1, \dots, n-1\}$ 에서 난수 값 k 를 생성한다.

- ② 타원곡선 점 $(x_1, y_1) = kG$ 를 계산한다.
- ③ $r = \overline{x_1} \bmod n$ 을 계산한다. ($\overline{x_1}$ 은 유한체 원소 x_1 을 정수로 변환한 것이다.)
- ④ $r = 0$ 이면, ① 단계로 되돌아간다.
- ⑤ 메시지 해시값 $e = \text{SHA1}(m)$ 을 계산한다.
- ⑥ $s = (k^{-1}(e + dr)) \bmod n$ 을 계산한다.
- ⑦ $s = 0$ 이면, ① 단계로 되돌아가고 0이 아니면 메시지 m 에 대한 서명은 (r, s) 이다.

서명 검증과정은 다음과 같다[27].

- ① 인증된 공개키 (E, G, n, Q) 를 얻는다.
- ② 메시지 해시값 $e = \text{SHA1}(m)$ 을 계산한다.
- ③ 중간 값 $u_1 = es^{-1} \bmod n$ 을 계산한다.
- ④ 중간 값 $u_2 = rs^{-1} \bmod n$ 을 계산한다.
- ⑤ 타원곡선 점 $(x_2, y_2) = u_1G + u_2Q$ 를 계산한다.
- ⑥ $t = \overline{x_2} \bmod n$ 을 계산한다.
- ⑦ $r = t$ 인지 확인한다.

※ 서명 생성 과정 ⑤단계와 서명 검증과정 ②단계의 SHA-1은 2010년 이후 사용제한을 권고하고 있고, SHA-224과 SHA-256 등으로 대체하여 사용하는 것을 권장하고 있다[25, 19].

제4절 ECDSA 하드웨어 구현 검증의 필요성

ECDSA는 앞서 설명한 바와 같이 타원곡선 알고리즘의 효율성으로 인하여 유한체 위에서의 타원 곡선을 하드웨어 적으로 구현하기 위한 노력이 이루어지고 있다. 그러나 타원곡선 암호 알고리즘에 대한 다양한 하드웨어 구조가 제안되고 있기 때문에 구현된 모듈이 제대로 동작하는지에 대한 평가가 매우 중요하다. 이에 따라 다양한 하드웨어 구조를 통해 구현된 암호 모듈을 평가해주는 제도로 암호모듈 검증 프로그램 CMVP(Cryptographic Module Validation Program)가 존재한다[20]. CMVP는 FIPS 140-2에 따라 암호알고리즘, 인증알고리즘, 서명알고리즘 등을 포함한 암호 모듈을 시험한다.

1. 암호 알고리즘 검증 방법

시험대상 구현물(Implementation Under Test, 이하 IUT)에 구현된 검증대상 암호알고리즘이 관련 표준에 따라 정확하게 구현되었음을 주장하기 위해서는 암호 알고리즘 검증기준에 포함된 검사를 성공적으로 수행하여야 한다. 이에 따라 NIST에서 제안한 암호 알고리즘을 검증하는 방법은 3가지로 나뉜다. 3가지는 Known Answer Test, Monte-Carlo Test, Modes Test이다.

1) Known Answer Test

Known Answer Test는 일정한 패턴의 값을 입력 값으로 하여 입력 값의 결과를 알고 있는 상태에서 테스트 하는 방법이다. 이 방법은 전체 암호 알고리즘의 검증과 암호 알고리즘의 하위 모듈의 테스트에서 사용한다. 입력 값은 평문을 고정하고 키 값을 일정한 규칙에 의해서 변화된 값을 가지고 테스트 하는 방법, 키를 고정하고 평문을 일정한 규칙에 의해서 변화된 값을 입력하여 그 결과 데이터와 비교 하여 검증하는 방법이 있다[1].

2) Modes Test

Modes Test는 Monte-Carlo Test의 근간이 되는 테스트 방법으로 테스트 하고자 하는 암호 알고리즘을 랜덤하게 설정된 입력 값을 입력하여 10000번 반복하는 동안 입력 값은 그 전 결과 값을 다시 입력하는 방식이다. 이렇게 10000번 반복한 결과를 비교 하며, 이 때 새로운 랜덤 값을 입력 하여 다시 위의 과정을 400번 반복하여 비교 검증하는 방법이다[1].

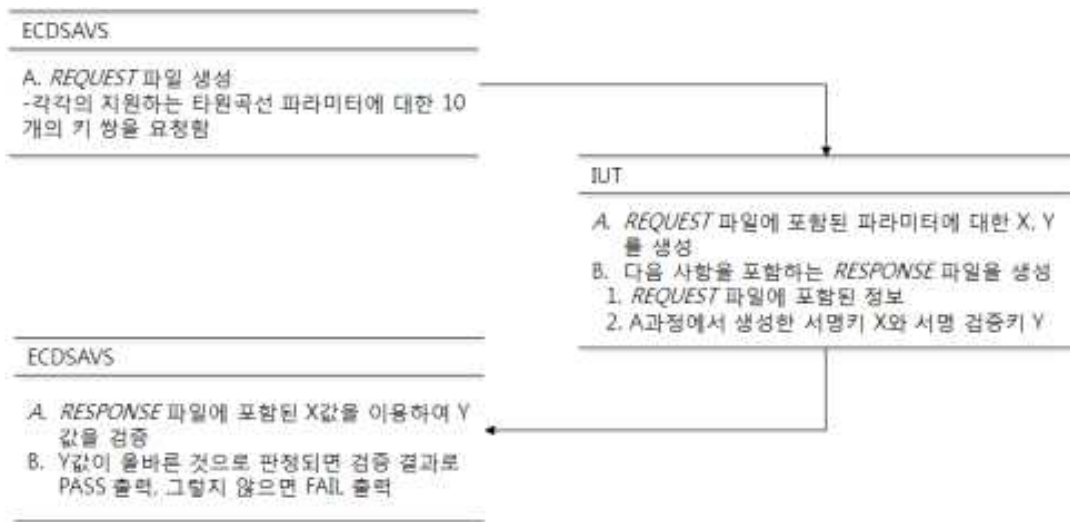
3) Monte-Carlo Test

Monte-Carlo Test는 난수발생기를 이용하여 생성된 테스트 벡터에 대하여 출력 값과 예상 출력 값에 대한 비교로 평가가 이루어진다. 하나의 테스트 벡터를 입력 값으로 하여 생성된 출력 값을 새로운 입력 값으로 사용하는 과정을 각 알고리즘 검증 과정에 제시된 횟수만큼 반복하여 생성된 출력 값을 검증 받는다. Monte-Carlo Test는 임의의 값을 사용함으로써 Known Answer Test에서 발견하지 못한 오류를 찾아내기 위한 테스트이다[11].

2. ECDSAVS

기존 ECDSA 검증시스템(Elliptic Curve Digital Signature Algorithm Validation System, 이하 ECDSAVS)은 키 생성 테스트, 공개키 검증 테스트, 서명 생성 테스트, 서명 검증 테스트 검사를 수행하며, FIPS PUB 186-3에서 정의된 타원곡선을 검증대상으로 한다.

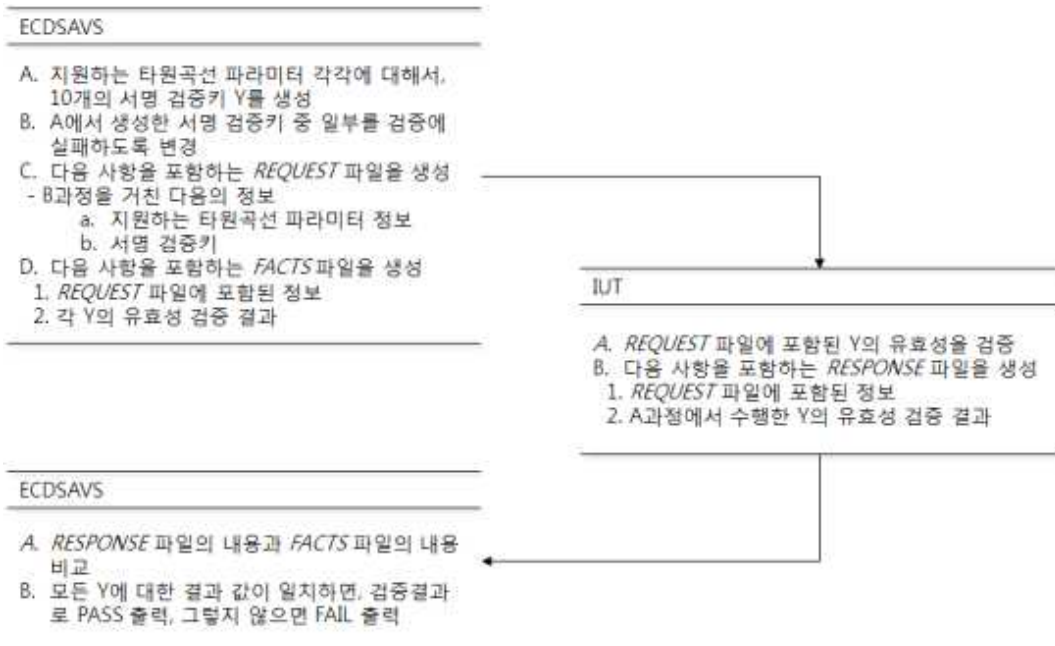
키 생성 테스트는 IUT가 서명키 X와 서명 검증키 Y를 올바르게 생성하는지를 검사한다. 이를 위해 ECDSAVS는 지원하는 타원곡선의 파라미터에 대한 서명키/서명 검증키(X, Y) 생성을 IUT에게 요청한다. IUT는 서명키/서명 검증키를 생성하고, ECDSAVS에게 전달한다. ECDSAVS는 전달받은 서명키/ 서명 검증키를 검증한다.



<그림 2-5> 키 생성 테스트 과정

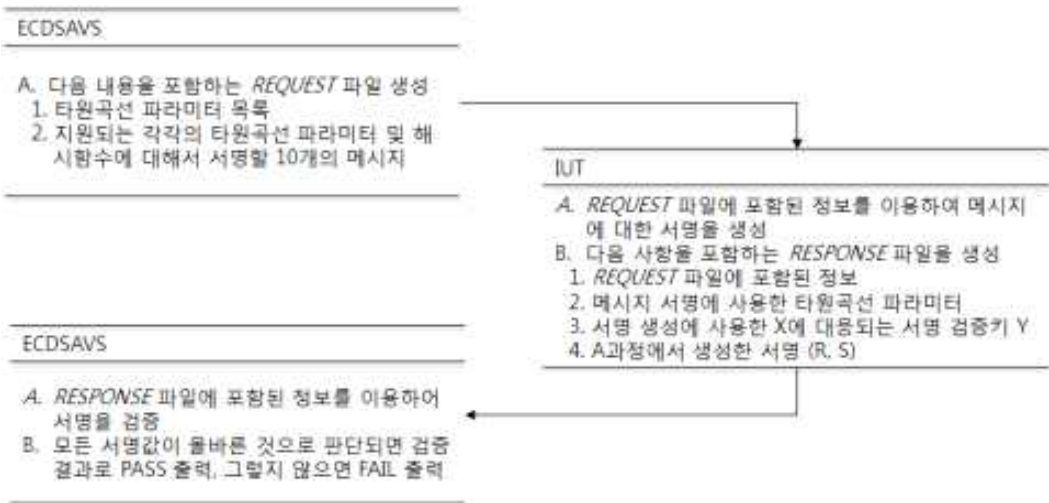
공개키 검증 테스트는 IUT가 ECDSA 전자서명을 위해 생성된 서명 검증키의 유효성을 판별할 수 있는지 검사한다.

이를 위해 ECDSAVS는 지원하는 타원곡선 파라미터 각각에 대해 10개의 서명 검증키를 생성하고 일부 서명 검증키를 변경하여 IUT에게 전달한다. IUT는 전달받은 서명 검증키의 유효성을 판별 후 그 결과를 ECDSAVS에게 전달하고, ECDSAVS는 IUT로부터 받은 결과를 자신이 알고 있는 값과 비교한다.



<그림 2-6> 공개키 검증 테스트 과정

서명 생성 테스트는 IUT가 서명을 올바르게 생성하는 지를 검사한다. 이를 위해 ECDSA는 지원하는 타원곡선 파라미터/해시함수 및 서명할 메시지를 IUT에게 전달한다. IUT는 메시지에 대한 서명 후 생성한 서명과 서명 검증키를 ECDSA에게 전달한다. ECDSA는 전달 받은 서명 검증키를 사용하여 서명을 검증한다.



<그림 2-7> 서명 생성 테스트 과정

서명 검증 테스트는 IUT가 서명 검증 과정을 통해 생성된 서명의 유효성을 판별할 수 있는지 검사한다. 이를 위해 ECDSAVS는 서명키/서명 검증키(X, Y) 그리고 메시지(M)에 대한 서명(R, S)를 생성 후 일부 서명 검증키, 메시지, 서명을 서명 검증이 실패하도록 변경하여 IUT에게 전달한다. IUT는 전달받은 서명을 검증 한 후 결과를 ECDSAVS에게 전달한다. ECDSAVS는 IUT로부터 받은 결과를 자신이 알고 있는 값과 비교한다.

ECDSA

- A. 지원하는 타원곡선 파라미터에 대해서 다음 정보를 갖는 15개의 데이터 집합을 생성
1. 지원하는 해시함수
 2. 타원곡선 파라미터에 부합하는 서명키/서명 검증키 쌍
 3. 메시지 M
 4. 서명키를 사용하여 생성한 메시지에 대한 서명 (R, S)
- B. 서명 검증키, 메시지, 서명으로 구성된 집합 중 서명 검증 과정이 실패하도록 서명 검증키, 메시지, 서명 중에 하나를 변경
- C. 다음 사항을 포함하는 *REQUEST* 파일을 생성
1. 타원곡선 파라미터
 2. 지원하는 해시함수
 3. B과정을 거친 다음의 정보
 - a. 메시지
 - b. 서명 검증키
 - c. 서명값
- D. 다음 사항을 포함하는 *FACTS* 파일을 생성
1. *REQUEST* 파일에 포함된 정보
 2. 서명 생성에 사용한 서명키
 3. B과정을 거친 각 메시지, 서명 검증키, 서명 데이터 집합에 대한 서명 검증 결과

IUT

- A. *REQUEST* 파일에 포함된 정보를 이용하여 서명을 검증
- B. 다음 사항을 포함하는 *RESPONSE* 파일을 생성
1. *REQUEST* 파일에 포함된 정보
 2. A과정에서 수행한 서명 검증 결과

ECDSA

- A. *RESPONSE* 파일과 *FACTS* 파일의 내용 비교
- B. *RESPONSE* 파일내의 서명 검증 결과가 *FACTS* 파일의 내용과 일치하면, 검증 결과로 PASS 출력, 그렇지 않으면 FAIL 출력

<그림 2-8> 서명 검증 테스트 과정

앞서 설명한 4가지의 검증 테스트와 같이 CMVP는 모듈 전체에 대한 입·출력 값의 검증을 하므로 검증이 실패하면 어떤 내부 모듈이 잘못되었는가에 대한 피드백은 없다. 이러한 문제점을 개선하기 위해서는 타원곡선 암호의 구현에 필요한 내부 모듈의 검증 과정이 필요하다. 따라서 ECDSA의 내부 모듈인 유한체의 사칙연산 모듈부터 스칼라 곱셈 모듈, ECDSA 서명 및 검증 모듈까지 기능 검증을 거칠 수 있는 S/W를 개발하였다.

제3장 ECDSA 하드웨어 기능 검증 소프트웨어

제1절 검증 소프트웨어 설계

본 절에서는 앞서 제기한 CMVP의 문제점에 대한 해결책으로 목표 하드웨어의 구조에 따라 암호 모듈 개발 과정에서부터 검증을 수행할 수 있도록 ECDSA 하드웨어 검증 소프트웨어 구조를 설계한다. 목표 하드웨어의 특징에 맞추어 소프트웨어를 구현함으로써 모듈 검증의 신뢰성을 높인다. 또한, 기존의 검증된 ECDSA 프로그램을 통해 ECDSA 하드웨어 검증 소프트웨어를 검증하였다.

1. 목표 ECDSA 파라미터

타원곡선 전자서명을 구현하기 앞서 필요한 파라미터 값은 다음과 같다 [27].

<표 3-1> 타원곡선 전자서명 파라미터 값

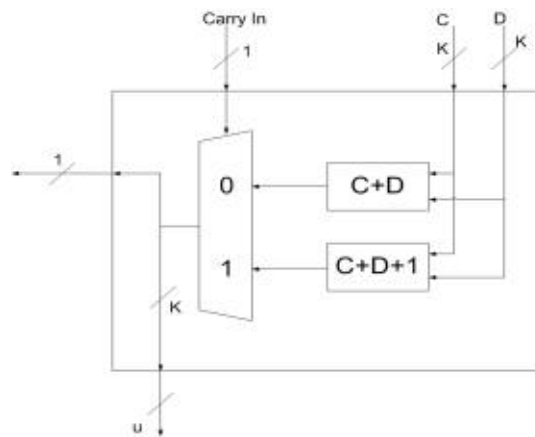
$p_{224} = 2^{224} - 2^{96} + 1$	FF00000000000000000000000001
$y^2 = x^3 + ax + b$	a : FFFE b : B4050A850C04B3ABF54132565044B0B7D7BFD8BA270B39432355FFB4
G	x : B70E0CBD6BB4BF7F321390B94A03C1D356C21122343280D6115C1D21 y : BD376388B5F723FB4C22DFE6CD4375A05A07476444D5819985007E34
n	FF16A2E0B8F03E13DD29455C5C2A3D

타원곡선 $E : y^2 = x^3 + ax + b$ 에서 a 값과 b 값과 소수 p 의 값, 생성원 G , 위수 n 의 값을 알아야 ECDSA를 구현할 수 있다. 본 소프트웨어는 224비트의 ECDSA를 선택하였고 이에 맞추어 소수체 $p_{224} = 2^{224} - 2^{96} + 1$ 를 이용한다.

2. 목표 하드웨어 기본 구조

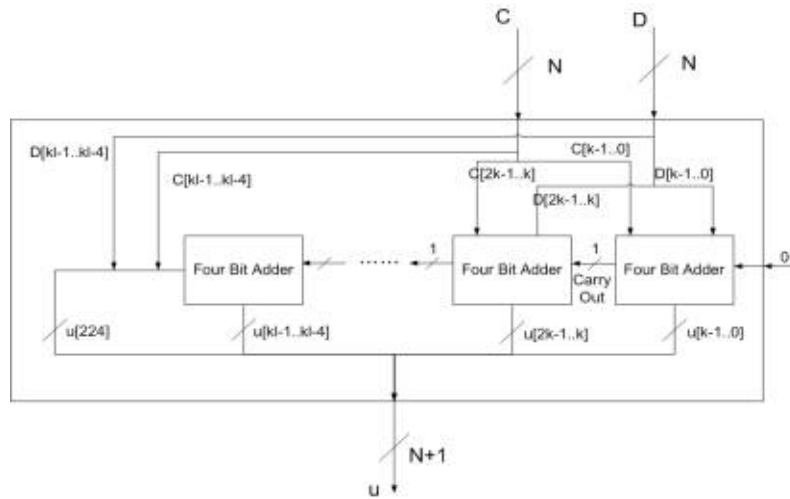
소프트웨어를 설계하기 앞서 목표 하드웨어의 구조에 대해 알아보아야 한다. 본 하드웨어의 특징은 4비트 단위 덧셈을 하며 한 비트 씩 순차적 곱셈, 오른쪽 비트부터 2배 연산과 덧셈을 반복하는 스칼라 곱셈이다. 이에 따라 하드웨어 구조를 나타내었다.

<그림 3-1>은 목표 하드웨어의 하위 모듈인 덧셈 내부 구조이다.



<그림 3-1> k 비트 단위 덧셈

224비트를 k비트 단위로 나누어 입력하면 덧셈을 한 후 캐리 값과 연산결과 값을 출력한다. 캐리 값은 상위 비트에 보내 캐리 값이 1이면 $C+D+1$ 연산을 선택하고, 0이면 $C+D$ 연산을 선택하도록 구현되었다. 이러한 과정은 캐리 값을 받아 선택하는 과정이 존재하지만 실제 내부의 $C+D+1$ 연산과 $C+D$ 연산은 병렬로 처리함으로써 하드웨어 적으로 구현하였을 때 속도의 향상을 기대할 수 있다. 따라서 k비트 단위의 병렬구조를 통해 총 224비트 값의 덧셈을 하도록 구현되었고 그 구조는 <그림 3-2>와 같다[8].



<그림 3-2> 224비트 덧셈기 하드웨어 구조

224비트 덧셈기 하드웨어 구조를 살펴보면 k 비트씩 쪼개어 Four Bit Adder 모듈을 병렬로 처리하고 출력되는 캐리 값을 통해 상위 비트에 보낸다. u 값은 k 비트씩 쪼개져 나온 결과를 다시 붙이고 최상위 비트의 캐리 값을 포함하여 총 225비트의 결과를 출력해준다.

다음으로 목표 하드웨어의 유한체 곱셈은

$$\begin{aligned}
 AB \bmod P &= A(b_{n-1}, b_{n-2}, \dots, b_1, b_0) \bmod P \\
 &= b_{n-1}(A2^{n-1}) + b_{n-2}(A2^{n-2}) + \dots + b_1(A2) + b_0A \bmod P \\
 &= (\dots((b_{n-1}A2 \bmod P) + b_{n-2}A \bmod P)2 \bmod P + \dots) + b_0A \bmod P
 \end{aligned}$$

로 계산될 수 있다. 계산 과정은 224비트의 A 와 B 를 입력하면 B 의 한 비트를 A 와 곱셈을 하는데 A 가 1인 경우에는 왼쪽으로 시프트 연산과 덧셈을 한 후 다시 반복되며, 0인 경우에는 덧셈을 한 후 반복된다. 이러한 과정을 통해 A 와 B 의 곱셈결과를 얻을 수 있다.

마지막으로 하드웨어 특징인 타원곡선 위의 점 연산을 오른쪽 비트부터 2배 연산을 반복하는 스칼라 곱셈식은 다음과 같다.

$$dP = (d_{n-1}, d_{n-2}, \dots, d_0)P$$

$$= (d_0P + d_12P + d_22(2P) + \dots)$$

소수체 GF(p) 위의 타원곡선 $E : y^2 = x^3 + ax + b$ 위의 점 P에 대해 스칼라 d로 곱셈을 하여 dP를 구하는 병렬 알고리즘을 다음과 같이 구할 수 있다.[28]

<표 3-2> 스칼라곱셈 알고리즘

스칼라 곱셈 알고리즘	
1.	input $P, d = (d_{n-1}, d_{n-2}, \dots, d_0)$
2.	$Q[0] \leftarrow O, Q[1] \leftarrow O, R[0] \leftarrow O, R[1] \leftarrow P$
3.	for i from 1 to n do
3.1.	$R[0] \leftarrow R[1]$
3.2.	$R[1] \leftarrow 2R[1]$
3.3.	$Q[1] \leftarrow Q[0] + R[0]$
3.4.	$Q[0] \leftarrow Q[d_{i-1}]$
4.	output $Q[0]$

검증과정에서 신뢰성을 얻기 위해서는 앞서 설명한 하드웨어 구조의 특징에 맞추어 소프트웨어를 구현하여야 한다. 또한, 이러한 특징 이외에도 하드웨어의 전체적인 구조도 동일하게 소프트웨어로 구현해야 한다.

3. ECDSA 하드웨어 맞춤형 소프트웨어 설계 및 구현

ECDSA 하드웨어를 검증하는 여러 방법을 살펴보면 하드웨어의 구현이 완료 되었을 때 상위 모듈을 검증한다. 그러나 앞서 제기했던 요구사항을 만족시키기 위해서는 하드웨어의 개발 과정에서부터 검증과정을 진행해야 ECDSA의 구현을 용이하게 할 수 있다.

목표 하드웨어 구조의 특징 중 하나인 k비트로 나누어 입력되는 방식에서 선택한 4비트 단위에 맞추어 검증 소프트웨어의 모든 모듈이 4비트 단위로 동작하도록 구현하였다. 224비트의 파라미터 값이 들어오면 char 형 변수에 4비트 단위로 나누어 들어오게 되는데 연산 과정에서 carry 값을 위해 상위 1비트를 추가로 사용하여 총 5비트를 사용한다. 연산과정이 끝나면 상위 1비트의 carry값은 삭제하는 과정을 거쳐서 4비트 단위로 출력한다.

본 소프트웨어는 프로그램의 테스트벡터 신뢰성을 높이기 위해 기존에 검증된 소프트웨어를 이용해 테스트 하였다. 기존 검증된 검증 소프트웨어는 스칼라 곱셈 모듈만을 검증하며, 내부 모듈은 검증하지 않는다. 따라서 본 소프트웨어는 스칼라 곱셈 아래 내부 모듈의 입·출력 값을 테스트벡터로 만들었으며, 상위 모듈인 타원곡선 전자서명까지 구현하였다.

내부 모듈 중 가장 많이 사용되며 기본적인 함수는 <표 3-3>과 같다. INTEGER_Adder 함수와 INTEGER_Sub함수는 유한체 원소의 덧셈과 뺄셈 모듈로서 앞서 설명한 224비트 덧셈기 하드웨어 구조를 적용하였다. 두 함수는 각각 테스트벡터를 제공하도록 구현되었다.

<표 3-3> 유한체 덧셈과 뺄셈 테스트벡터 제공 함수

```
void INTEGER_Adder()
void INTEGER_Sub()
```

앞서 구현된 모듈을 이용하여 곱셈, 역원, 모듈로 연산이 가능하며, 이 연산 모듈의 테스트벡터 제공 함수는 <표 3-4>이다.

<표 3-4> 유한체 곱셈과 역원 및 모듈러 테스트벡터 제공 함수

```
void MOD_UmodP()
void MOD_Multi_modP()
void MOD_Invertor_ModP()
void MOD_UmodN()
void MOD_Multi_modN()
void MOD_Invertor_ModN()
```

하위 모듈을 이용해 타원곡선 점 연산을 구현할 수 있으며 이에 따라 테스트백터를 제공하는 함수는 <표 3-5>이다. 또한, 타원곡선 위의 두 점 덧셈과 한 점의 두배 연산을 이용해 스칼라 곱셈을 구현할 수 있다.

<표 3-5> 타원곡선 위의 점 연산 테스트백터 제공 함수

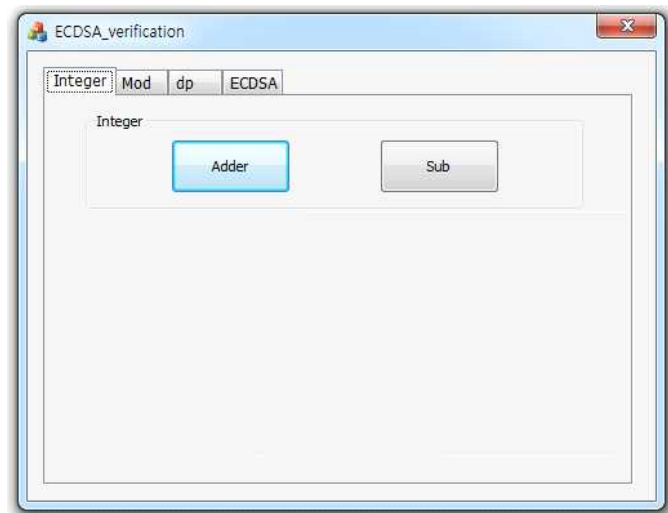
```
void DP_dp_2P()
void DP_dp_module()
void DP_dp_PplusQ()
```

다음으로 ECDSA 전자서명의 서명과 검증 함수를 통해 서명 생성과 서명 검증 테스트백터를 제공한다.

<표 3-6> ECDSA 전자서명 테스트백터 제공 함수

```
void ECDSA_ECDSA_Sign()
void ECDSA_ECDSA_VERIFY()
```

검증 프로그램은 C++로 구현하였으며 아래 <그림 3-3>은 프로그램의 초기화면이다. 이 프로그램은 앞서 설명한 함수를 이용해 내부 모듈을 검증할 수 있는 테스트 백터를 제공하도록 구성되어있고, 일부 많은 연산과정을 거치는 모듈은 연산 중간 값을 텍스트파일로 제공한다.

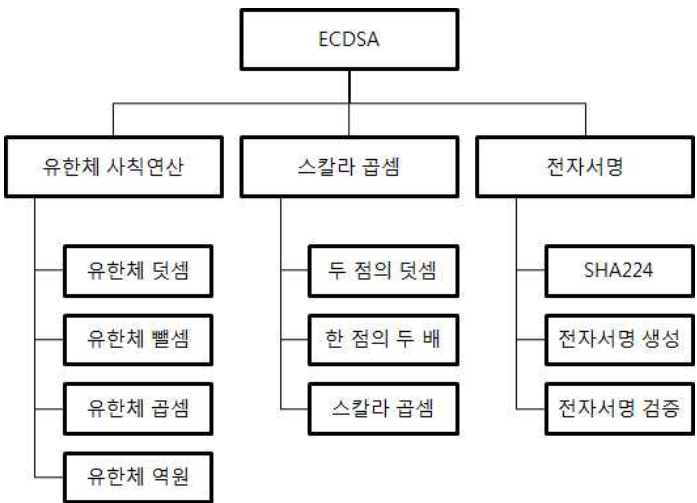


<그림 3-3> ECDSA 하드웨어 검증 소프트웨어 초기화면

제2절 ECDSA 하드웨어 내부 모듈 검증

타원곡선 암호 알고리즘을 하드웨어로 구현할 때는 유한체의 덧셈, 뺄셈, 곱셈, 역원이 내부 모듈로 필요하다. 이러한 유한체 사칙연산 모듈을 이용하여 상위 모듈인 타원곡선 위의 점 연산 모듈이 필요하며, 점 연산과정은 두 점의 덧셈과 한 점의 두 배 연산이 있다. 타원곡선 위의 점 연산을 통해 스칼라 곱셈을 구현할 수 있다. 따라서 전체 내부 모듈을 이용해 타원곡선 암호 알고리즘을 구현할 수 있다.

본 절에서는 앞서 설명한 하드웨어 구현 검증의 필요성에 따라 타원곡선 전자서명의 내부 모듈인 유한체 사칙연산부터 타원곡선 점 연산, 스칼라 곱셈, 전자서명 서명생성 및 서명검증 모듈의 검증과정을 거친다.



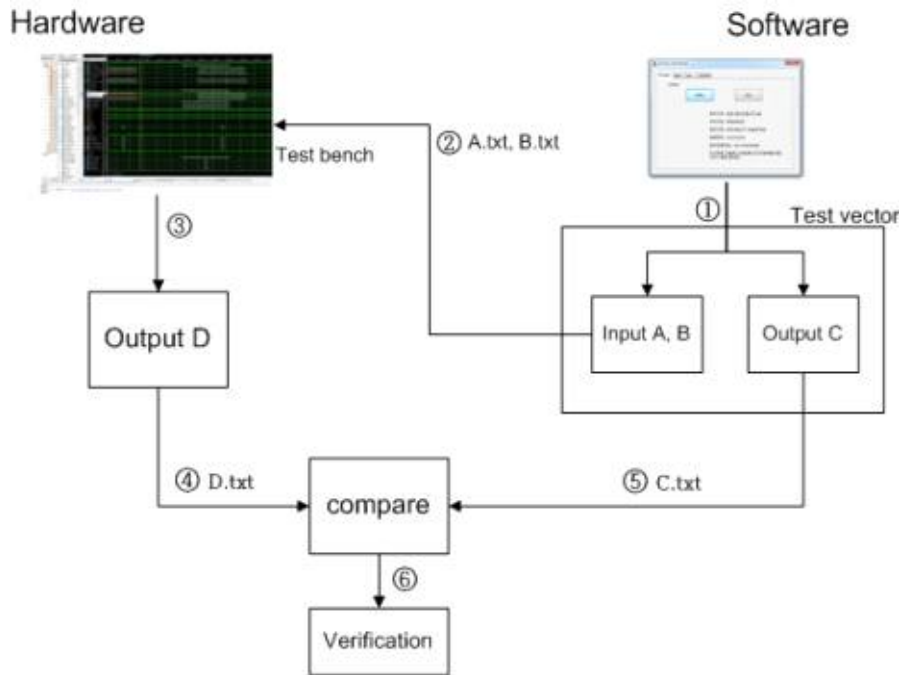
<그림 3-4> ECDSA 내부 모듈

1. 하드웨어 검증 방법

검증 모듈 설명에 앞서 1회 검증방식의 검증 순서는 다음과 같다.

- ① 검증 소프트웨어에서 해시함수를 이용해 난수를 생성하여 사용되는 입력 값 A와 B를 각 모듈에 따라 동작 후 결과 값 C를 생성한다.

- ② 입력 값 테스트벡터 A.txt와 B.txt를 하드웨어에 전달한다.
 - ③ 하드웨어에서 테스트벡터 A.txt와 B.txt를 가지고 모듈을 동작시켜 결과 값 D를 생성한다.
 - ④ 하드웨어 결과 값 D를 D.txt 파일로 받는다.
 - ⑤ 소프트웨어 결과 값 C를 C.txt 파일로 받는다.
 - ⑥ 하드웨어 결과 값과 D와 소프트웨어 결과 값 C를 비교하여 검증한다.
- 이 순서를 여러번 반복해 테스트하여 하드웨어 내부 모듈을 검증할 수 있다.



<그림 3-5> ECDSA 하드웨어 검증 순서

2. 유한체 사칙연산 모듈 검증

ECDSA의 하위 모듈인 유한체 사칙연산 모듈 검증은 정수형 덧셈과 뺄셈 뿐만 아니라, 모듈러 연산을 포함하여 총 10가지의 모듈이 존재하며 다음과 같다.

- Adder
- Sub
- UmodP
- CplusDmodP
- UsubVmodP
- UmodN
- Multiplier_ModP
- Invertor_modP
- Invertor_modN
- Multiplier_modN

10가지 모듈 중 모듈러 연산은 modP와 modN으로 나뉜다. modP는 스칼라 곱셈까지 연산과정에서 필요하고 modN은 전자서명 과정에서 필요하다.

Adder 모듈은 앞서 설명한 하드웨어 구조에 맞게 4비트 단위의 덧셈을 반복하여 224비트의 정수 덧셈을 구현하였다. 두 원소의 값 a , b 와 덧셈 결과는 <표 3-7>와 같으며, 정수 덧셈 모듈의 검증 소프트웨어 화면은 <그림 3-6>이다.

<표 3-7> Adder 테스트벡터 예

a	0482580A0EC5BC47E88BC8C378632CD196CB3FA058A7114EB03054C9
b	AE99FEEBB5D26945B54892092A8AEE02912930FA41CD114E40447301
덧셈 결과	0B31C56F5C498258D9DD45ACCA2EE1AD427F4709A9A74229CF074C7CA

Adder

Adder

A 0482580A0EC5BC47E88BC8C378632CD196CB3FA058A7114EB03054C9

B AE99FEEB85D26945B54892092A8AEE02912930FA41CD114E40447301

결과 0B31C56F5C498258D9DD45ACCA2EE1AD427F4709A9A74229CF074C7CA

Start

설명

224 bit의 A값과 224 bit의 B값을 16진수로 입력하여
A + B 의 결과를 출력해준다.

확인

<그림 3-6> Adder 모듈 검증

Sub 모듈은 Adder를 이용해 구현된다. $a + \sim b + 1$ 을 통해 정수형 뺄셈결과를 얻을 수 있다. 뺄셈의 두 원소의 값 a , b 와 뺄셈 결과는 <표 3-8>와 같으며, 뺄셈 모듈의 검증 소프트웨어 화면은 <그림 3-7>이다.

<표 3-8> Sub 테스트벡터 예

a	B8B50B7EA73167B36397172354E048CEC33C07C3EC2E704AD4D0296B
b	A9AAB6E39F1E367AE8BDA7CB2FD66CDB8958E8EC7AD9D9ACAE7D4FDB
뺄셈 결과	00F0A549B081331387AD96F582509DBF339E31ED77154969E2652D990

sub

Sub

A B8B50B7EA73167B36397172354E048CEC33C07C3EC2E704AD4D0296B

B A9AAB6E39F1E367AE8BDA7CB2FD66CDB8958E8EC7AD9D9ACAE7D4FDB

결과 00F0A549B081331387AD96F582509DBF339E31ED77154969E2652D990

Start

설명

224 bit의 A값과 224 bit의 B값을 16진수로 입력하여
A - B 의 결과를 출력해준다.

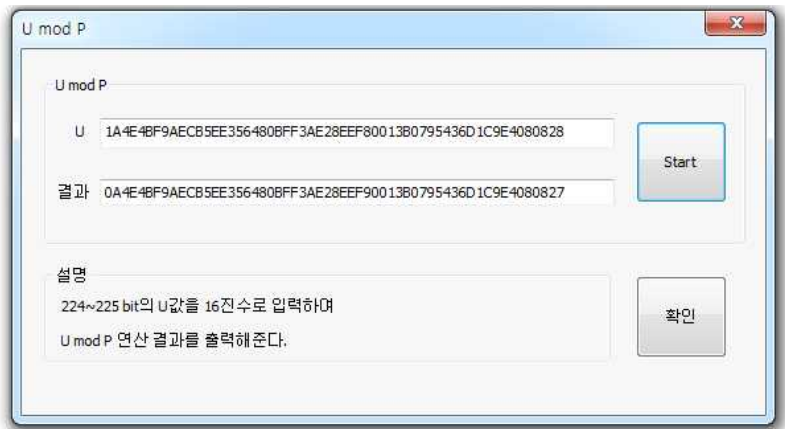
확인

<그림 3-7> Sub 모듈 검증

UmodP 모듈은 Adder를 이용해 구현된다. 모듈러 p 연산의 한 원소 값 u 와 그 결과는 <표 3-9>와 같으며, UmodP 모듈의 검증 소프트웨어 화면은 <그림 3-8>이다.

<표 3-9> UmodP 테스트벡터 예

u	1A4E4BF9AECB5EE356480BFF3AE28EEF80013B0795436D1C9E4080828
결과	0A4E4BF9AECB5EE356480BFF3AE28EEF90013B0795436D1C9E4080827

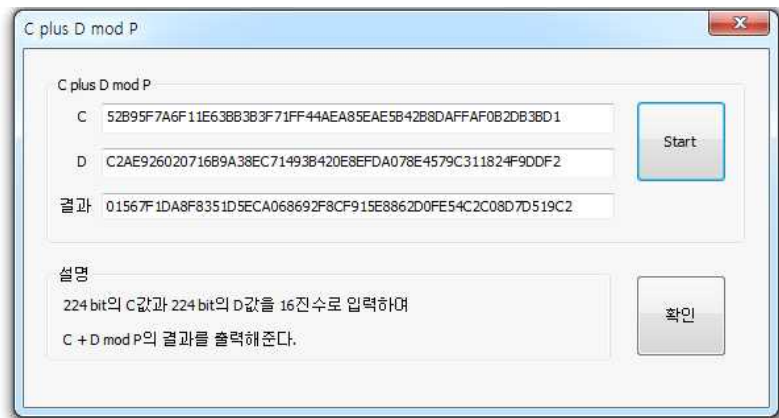


<그림 3-8> UmodP 모듈 검증

CplusDmodP 모듈은 유한체 덧셈 연산 모듈로서 Adder와 UmodP를 이용해 구현된다. 유한체 덧셈의 두 원소의 값 c , d 와 덧셈 결과는 <표 3-10>와 같으며, CplusDmodP 모듈의 검증 소프트웨어 화면은 <그림 3-9>이다.

<표 3-10> CplusDmodP 테스트벡터 예

c	52B95F7A6F11E63BB3B3F71FF44AEA85EAE5B42B8DAFFAF0B2DB3BD1
d	C2AE926020716B9A38EC71493B420E8EFDA078E4579C311824F9DDF2
덧셈 결과	01567F1DA8F8351D5ECA068692F8CF915E8862D0FE54C2C08D7D519C2

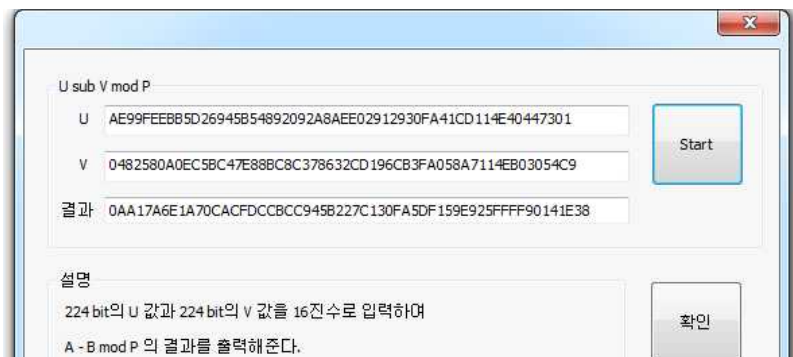


<그림 3-9> CplusDmodP 모듈 검증

UsubVmodP 모듈은 유한체 뺄셈 모듈로서 Adder와 UmodP를 이용해 구현된다. $U + (\sim V + t) + 1$ 을 통해 유한체 뺄셈과 같은 결과를 얻을 수 있다. 유한체 뺄셈의 두 원소의 값 a , b 와 뺄셈 결과는 <표 3-11>와 같으며, 유한체 뺄셈 모듈의 검증 소프트웨어 화면은 <그림 3-10>이다.

<표 3-11> UsubVmodP 테스트벡터 예

u	AE99FEEBB5D26945B54892092A8AEE02912930FA41CD114E40447301
v	0482580A0EC5BC47E88BC8C378632CD196CB3FA058A7114EB03054C9
뺄셈 결과	0AA17A6E1A70CACFDCCBCC945B227C130FA5DF159E925FFFF90141E38

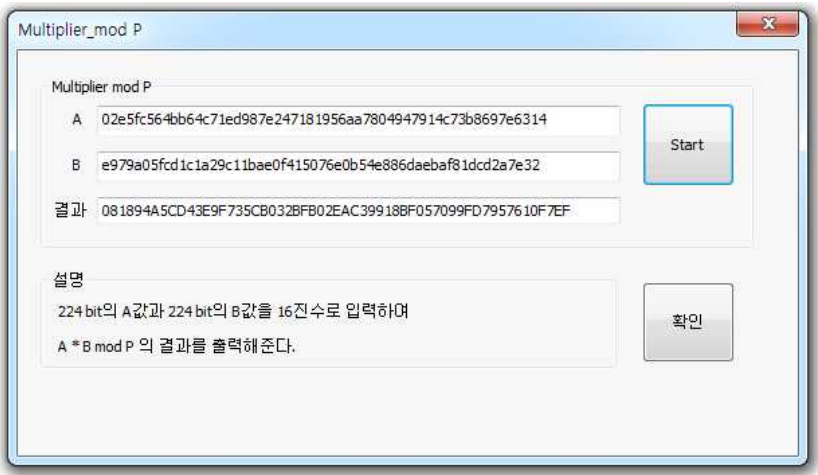


<그림 3-10> UsubVmodP 모듈 검증

Multiplier_modP 모듈은 Adder와 UmodP를 이용해 구현된다. 한 비트씩 곱셈을 통해 유한체 곱셈결과를 얻을 수 있다. 유한체 곱셈의 두 원소의 값 a , b 와 곱셈 결과는 <표 3-12>와 같으며, 유한체 곱셈 모듈의 검증 소프트웨어 화면은 <그림 3-11>이다. 유한체 사칙연산의 모듈 종류가 다양하여 하드웨어 모듈 검증은 대표적으로 유한체 곱셈 모듈을 검증하였으며, 이 테스트백터를 하드웨어 모듈에 제공하여 시뮬레이션 결과를 얻은 것이 <그림 3-12>이다.

<표 3-12> Multiplier_modP 테스트백터 예

a	02e5fc564bb64c71ed987e247181956aa7804947914c73b8697e6314
b	e979a05fcd1c1a29c11bae0f415076e0b54e886daebaf81dcd2a7e32
곱셈 결과	81894A5CD43E9F735CB032BFB02EAC39918BF057099FD7957610F7EF



<그림 3-11> Multiplier_modP 모듈 검증

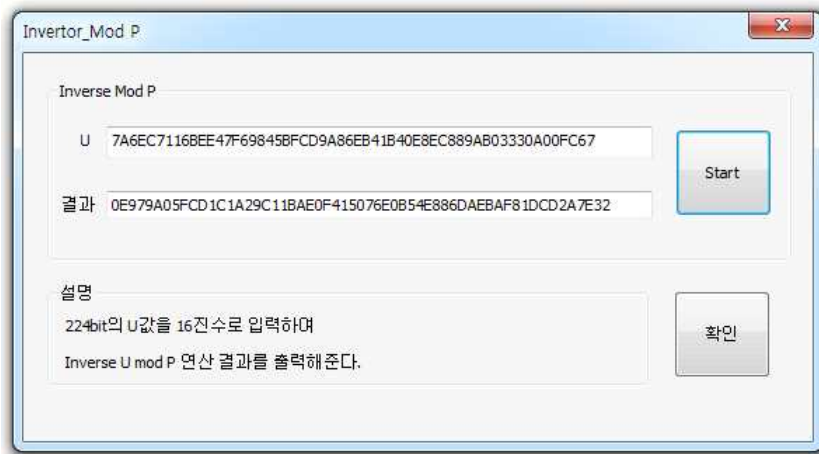


<그림 3-12> Multiplier_modP 하드웨어 시뮬레이션 결과

Invertor_modP 모듈은 Adder, UmodP, UsubV와 한 비트 시프트 연산을 이용해 구현된다. 유한체 한 원소 값 u 와 역원 결과는 <표 3-13>와 같으며, Invertor_modP 모듈의 검증 소프트웨어 화면은 <그림 3-13>이다.

<표 3-13> Invertor_modP 테스트벡터 예

u	7A6EC7116BEE47F69845BFCD9A86EB41B40E8EC889AB03330A00FC67
결과	0E979A05FCD1C1A29C11BAE0F415076E0B54E886DAEBAF81DCD2A7E32

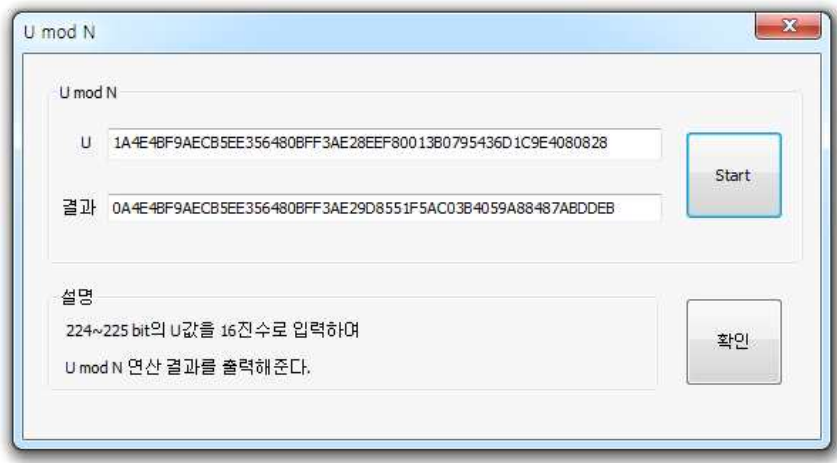


<그림 3-13> Invertor_modP 모듈 검증

다음은 ECDSA 전자서명의 서명 및 검증과정에서 모듈러 N 연산이 필요하므로 구현한 UmodN 모듈이다. UmodN은 Adder를 이용해 구현된다. 모듈러 N 연산의 한 원소 값 u 와 그 결과는 <표 3-14>와 같으며, UmodP 모듈의 검증 소프트웨어 화면은 <그림 3-14>이다.

<표 3-14> UmodN 테스트벡터 예

u	1A4E4BF9AECB5EE356480BFF3AE28EEF80013B0795436D1C9E4080828
결과	0A4E4BF9AECB5EE356480BFF3AE28EEF90013B0795436D1C9E4080827



<그림 3-14> UmodN 모듈 검증

Invertor_modN 모듈과 Multiplier_modN 모듈은 앞서 구현된 Invertor_modP 모듈과 Multiplier_modP 모듈과 비슷한 구조이기 때문에 테스트벡터 제공 방식이 같다.

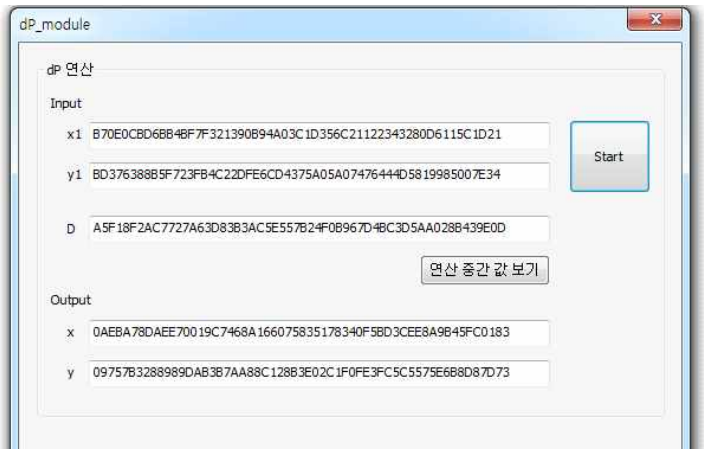
3. 스칼라 곱셈 모듈 검증

스칼라 곱셈 모듈은 타원곡선 점 연산 모듈과 하위 모듈을 이용해 타원곡선 위의 점 덧셈과 두 배를 반복하여 구현할 수 있다. 또한 스칼라 곱셈은 많은 연산과정을 거치므로 연산 중간 값을 확인할 수 있도록 구현하였다.

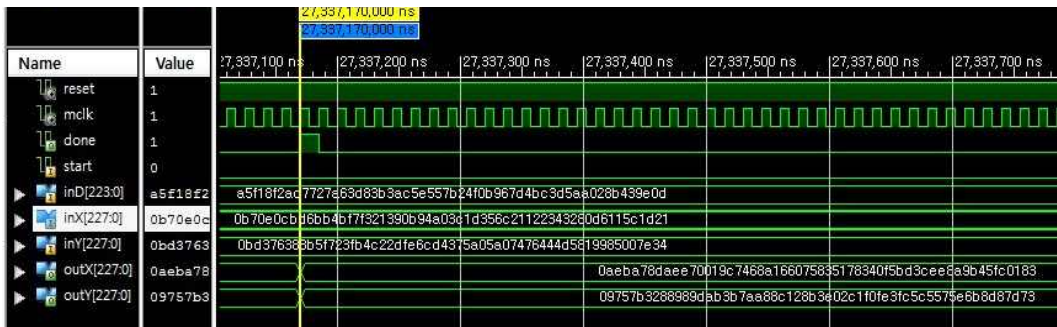
스칼라 곱셈 모듈에 필요한 점 $P(x,y)$ 와 224비트 원소 d 의 입력 값과 연산된 dP 의 결과 값은 <표 3-15>과 같다. 스칼라 곱셈 모듈의 검증 소프트웨어 화면은 <그림 3-15>이며, 소프트웨어에서 얻은 테스트벡터를 이용한 하드웨어 시뮬레이션은 <그림 3-16>이다.

<표 3-15> dP 테스트벡터 예

d	A5F18F2AC7727A63D83B3AC5E557B24F0B967D4BC3D5AA028B439E0D
$P(x,y)$	x : B70E0CBD6BB4BF7F321390B94A03C1D356C21122343280D6115C1D21
	y : BD376388B5F723FB4C22DFE6CD4375A05A07476444D5819985007E34
dP 결과	x : 0AEBA78DAEE70019C7468A166075835178340F5BD3CEE8A9B45FC0183
	y : 09757B3288989DAB3B7AA88C128B3E02C1F0FE3FC5C5575E6B8D87D73

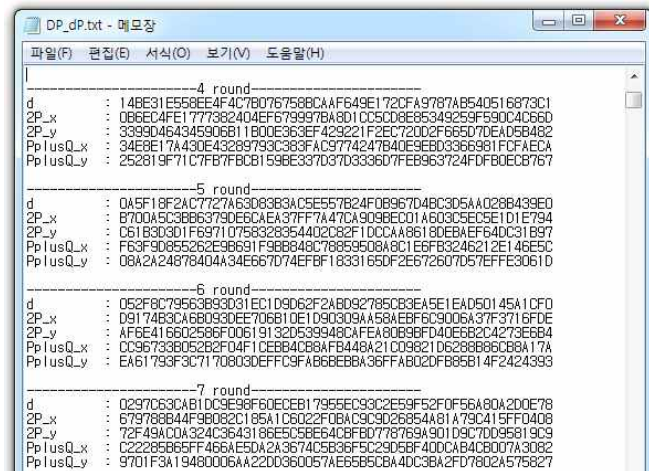


<그림 3-15> dP 모듈 검증



<그림 3-16> dP 하드웨어 시뮬레이션 결과

dP 연산 모듈은 224비트의 d값을 곱셈해야 하므로 224번 반복하여 곱셈을 한다. 앞서 설명한 특징대로 오른쪽 비트부터 두배 연산과 덧셈을 반복하여 구한다. 따라서 224번의 연산라운드를 텍스트파일에 저장하여 중간 값을 검증할 수 있도록 하였다.



<그림 3-17> dP 모듈 중간 값 검증

4. 타원곡선 점 연산 모듈 검증

앞서 스칼라 곱셈 모듈이 검증에 실패 했을 때에는 내부 모듈인 타원곡선 점 연산 모듈을 검증해보아야 한다. 타원곡선 점 연산 모듈은 유한체 사칙 연산 모듈을 이용해 구현가능하다. 타원곡선 점 연산은 타원곡선 위의 두

점 $P(x,y)$ 와 $Q(x,y)$ 을 이용한 두 가지 연산이 있으며 다음과 같다.

- $2P$ 연산
- $P + Q$ 연산

두 가지 연산은 내부 모듈인 유한체 사칙연산을 이용하여 구현되며 많은 연산과정을 거치므로 연산 중간 값을 확인할 수 있도록 구현하였다.

첫 번째로 $2P$ 연산 모듈은 유한체 사칙연산의 모든 모듈을 이용해 구현 된다. $2P$ 연산에 필요한 점 $P(x,y)$ 의 값과 그 결과는 <표 3-16>와 같으며, $2P$ 연산 모듈의 검증 소프트웨어 화면은 <그림 3-18>이다.

<표 3-16> $2P$ 테스트벡터 예

$P(x,y)$	x : B70E0CBD6BB4BF7F321390B94A03C1D356C21122343280D6115C1D21
	y : BD376388B5F723FB4C22DFE6CD4375A05A07476444D5819985007E34
$2P$ 결과	x : 0706A46DC76DCB76798E60E6D89474788D16DC18032D268FD1A704FA6
	y : 01C2B76A7BC25E7702A704FA986892849FCA629487ACF3709D2E4E8BB



<그림 3-18> $2P$ 모듈 검증

2장에서 본 바와 같이 $2P$ 연산의 식은

$$2P = (x_3, y_3), x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1,$$

$$y_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 (x_1 - x_3) - y_1$$

이다. 이 식에 맞게 구현된 하드웨어 $2P$ 연산 모듈의 계산 순서대로 중간 값을 텍스트파일에 출력하므로 연산과정을 검증할 수 있다. 먼저 유한체 곱셈과 덧셈을 이용해 $3x_1^2 + a$ 를 구한다. 다음으로 유한체 덧셈과 역원을 이용해 $\frac{1}{2y_1}$ 를 계산한다. 앞서 구한 $3x_1^2 + a$ 과 $\frac{1}{2y_1}$ 을 곱셈 모듈을 이용해 곱하면

$$\frac{3x_1^2 + a}{2y_1} \quad (3.1)$$

이다. 여기서 제곱을 하는 연산은 곱셈 모듈을 이용해 가능하며 제곱하면

$$\left(\frac{3x_1^2 + a}{2y_1}\right)^2 \quad (3.2)$$

이다. 식(3.2)에서 $2x_1$ 을 빼면 x_3 를 구할 수 있다.

$$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1 \quad (3.3)$$

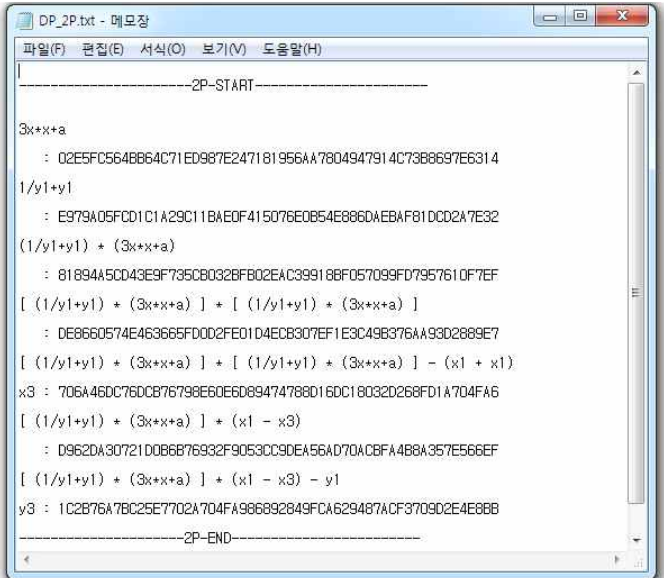
y_3 을 구하기 위해서 $(x_1 - x_3)$ 를 유한체 뺄셈을 통해 구한 후 앞서 구한 식 (3.1)에 곱하면

$$\left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) \quad (3.4)$$

이다. 식(3.4)에 y_1 을 빼면 y_3 을 구할 수 있다.

$$y_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) - y_1 \quad (3.5)$$

앞서 설명한 계산 순서에 따라 구현하였고, 제공되는 테스트벡터는 다음과 같다.



```

DP_2P.txt - 메모장
-----2P-START-----

3x+x+a
: 02E5FC564BB64C71ED987E247181956AA7804947914C73B8697E6314

1/y1+y1
: E979A05FCD1C1A29C11BAEDF415076E0B54E886DAEBAF81DCD2A7E32

(1/y1+y1) * (3x+x+a)
: 81894A5CD43E9F735CB032BF802EAC39918BF057099FD7957610F7EF

[ (1/y1+y1) * (3x+x+a) ] * [ (1/y1+y1) * (3x+x+a) ]
: DE8660574E463665FD0D2FE01D4ECB307EF1E3C49B376AA93D2889E7

[ (1/y1+y1) * (3x+x+a) ] * [ (1/y1+y1) * (3x+x+a) ] - (x1 + x1)
x3 : 706A46DC76DCB76798E60E6D69474788D16DC18032D268FD1A704FA6

[ (1/y1+y1) * (3x+x+a) ] * (x1 - x3)
: D962DA30721D0B6B76932F9053CC9DEA56AD70ACBFA4B8A357E566EF

[ (1/y1+y1) * (3x+x+a) ] * (x1 - x3) - y1
y3 : 1C2B76A7BC25E7702A704FA986892849FCA629487ACF3709D2E4E8BB

-----2P-END-----
  
```

<그림 3-19> 2P 모듈 중간 값 검증

다음은 $P+Q$ 연산 모듈이며 유한체 사칙연산의 모든 모듈을 이용해 구현된다. $P+Q$ 연산에 필요한 점 $P(x,y)$, $Q(x,y)$ 의 값과 그 결과는 <표 3-17>와 같으며, $P+Q$ 연산 모듈의 검증 소프트웨어 화면은 <그림 3-20>이다.

<표 3-17> $P+Q$ 테스트벡터 예

$P(x,y)$	x : AE99FEEBB5D26945B54892092A8AEE02912930FA41CD114E40447301
	y : 0482580A0EC5BC47E88BC8C378632CD196CB3FA058A7114EB03054C9
$Q(x,y)$	x : B70E0CBD6BB4BF7F321390B94A03C1D356C21122343280D6115C1D21
	y : BD376388B5F723FB4C22DFE6CD4375A05A07476444D5819985007E34
$P+Q$ 결과	x : 031C49AE75BCE7807CDFF22055D94EE9021FEDBB5AB51C57526F011AA
	y : 027E8BFF1745635EC5BA0C9F1C2EDE15414C6507D29FFE37E790A079B



<그림 3-20> $P + Q$ 모듈 검증

2장에서 본 바와 같이 $P + Q$ 연산의 식은

$$P + Q = (x_3, y_3), x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1} \right)^2 - x_1 - x_2,$$

$$y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1} \right) (x_1 - x_3) - y_1$$

이다. 이 식에 맞추어 $P + Q$ 연산 모듈을 하드웨어로 구현하였을 때 이에 맞추어 검증 소프트웨어를 구현한다. 검증 소프트웨어는 계산 순서대로 중간 값을 텍스트파일에 출력하고 하드웨어에 제공함으로써 중간 연산과정을 검증할 수 있다. $P + Q$ 연산 모듈의 계산순서는 다음과 같다.

먼저 $x_3 - x_1$ 를 유한체 뺄셈을 이용하여 구한 후 역원을 하면

$$\frac{1}{x_3 - x_1} \tag{3.5}$$

와 같다. 다음으로 $y_3 - y_1$ 을 계산 후 식(3.5)에 곱하면

$$\frac{y_3 - y_1}{x_3 - x_1} \quad (3.6)$$

이다. 식(3.6)을 제곱하면

$$\left(\frac{y_3 - y_1}{x_3 - x_1}\right)^2 \quad (3.7)$$

과 같다. 이어서 식(3.7)에 $(x_1 + x_3)$ 를 빼면 x_4 를 구할 수 있다.

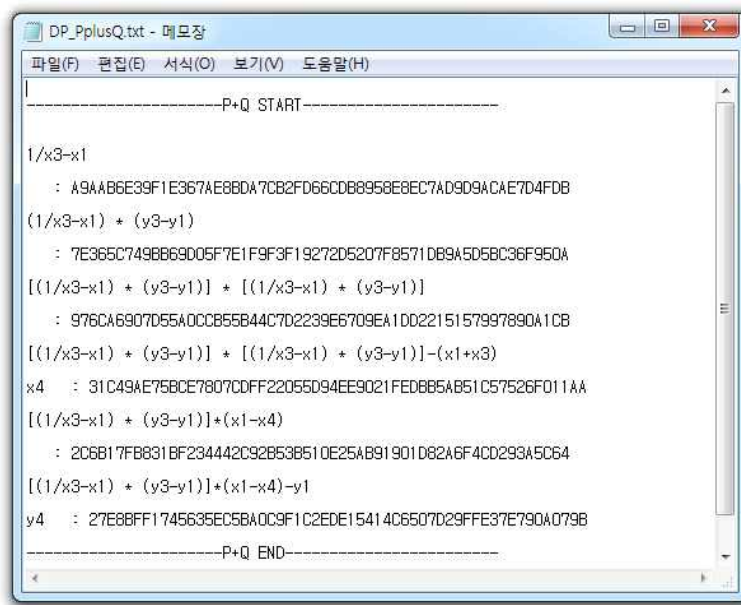
$$x_4 = \left(\frac{y_3 - y_1}{x_3 - x_1}\right)^2 - (x_1 + x_3) \quad (3.8)$$

앞서 구했던 식(3.6)에 $(x_1 - x_4)$ 를 곱하면

$$\left(\frac{y_3 - y_1}{x_3 - x_1}\right)(x_1 - x_4) \quad (3.9)$$

이다. 식(3.9)에 y_1 을 빼면 y_4 를 구할 수 있다.

$$y_4 = \left(\frac{y_3 - y_1}{x_3 - x_1}\right)(x_1 - x_4) - y_1 \quad (3.10)$$



<그림 3-21> P+Q 모듈 중간 값 검증

5. SHA-1 모듈 검증

SHA224모듈은 타원곡선 전자서명의 서브 모듈로 서명 생성과정과 검증 과정에서 필요하다. 따라서 SHA224 모듈에 필요한 메시지와 해시함수를 취한 결과 값은 <표 3-18>와 같으며, SHA224 모듈의 검증 소프트웨어 화면은 <그림 3-22>이다.

<표 3-18> SHA224 테스트벡터 예

m	ECDSA
결과	075B4574E544F9EA8378AF1E4E35A023C1AAB69FE9C50D89CE6FCF581



<그림 3-22> SHA224 모듈 검증

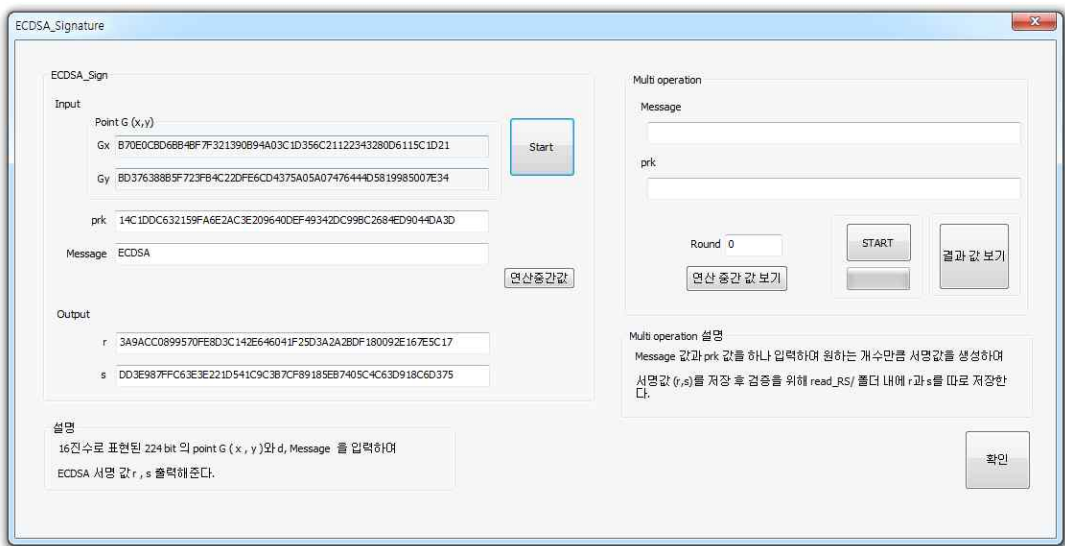
6. ECDSA 서명 모듈 검증

앞서 설명한 내부모듈이 완성되면 ECDSA 서명 모듈이 구현 가능하다. 서명을 구현할 때에는 모듈러 N 연산이 중간 과정에서 필요하다. 따라서 $UmodN$, $Invertor_modN$, $Multiplier_modN$ 모듈을 이용한다.

ECDSA 서명 테스트 벡터와 서명 결과 r, s 는 <표 3-19>와 같으며, 서명 모듈의 검증 소프트웨어 화면은 <그림 3-23>이다.

<표 3-19> ECDSA 서명 테스트벡터 예

$G(x, y)$	x : B70E0CBD6BB4BF7F321390B94A03C1D356C21122343280D6115C1D21
	y : BD376388B5F723FB4C22DFE6CD4375A05A07476444D5819985007E34
d	14C1DDC632159FA6E2AC3E209640DEF49342DC99BC2684ED9044DA3D
k	14C1DDC632159FA6E2AC3E209640DEF49342DC99BC2684ED030606A4
m	ECDSA
결과	r : 3A9ACC0899570FE8D3C142E646041F25D3A2A2BDF180092E167E5C17
	s : DD3E987FFC63E3E221D541C9C3B7CF89185EB7405C4C63D918C6D375



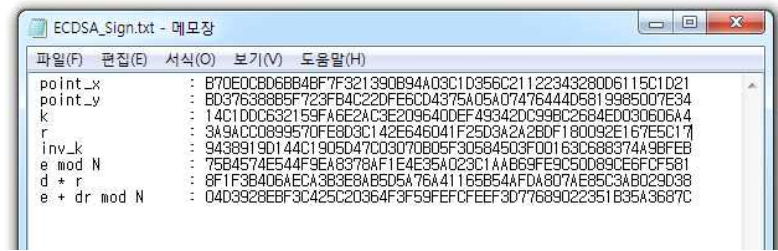
<그림 3-23> ECDSA 서명 모듈 검증

2장에서 본 바와 같이 서명 생성과정은 다음과 같다. 먼저 타원곡선 점 $(x_1, y_1) = kG$ 를 계산한다. 계산한 x_1 을 $\overline{x_1 \bmod n}$ 연산을 하면 r 을 구할 수 있다. 다음으로 s 를 구하기 위해 k^{-1} 과 같이 k 의 역원을 계산하고 메시지 해시값 $e = \text{SHA1}(m)$ 을 구한 후 $e \bmod N$ 연산을 한다. 이어서 $dr \bmod N$ 을 계산 후 e 와 더하면 다음과 같다.

$$(e + dr) \bmod n \quad (3.11)$$

앞서 계산한 k^{-1} 을 식(3.11)과 곱하여 s 를 구한다.

$$s = (k^{-1}(e + dr)) \bmod n \quad (3.12)$$



<그림 3-24> ECDSA 서명 중간 값

앞서 생성한 검증 소프트웨어의 테스트벡터를 이용해 하드웨어 서명 모듈을 시뮬레이션 하였고 그 결과는 <그림 3-25>과 같다. 시뮬레이션 결과를 보면 내부 중간 값을 볼 수 있는데 이 값과 테스트벡터의 일치 여부를 확인하여 서명 생성 연산과정에서 오류를 찾기가 쉽다.

Name	28,000 us	28,200 us	28,400 us	28,600 us	28,800 us
rstb					
k_inv[227:0]	09488919d144c1905d47c03070b05f305845d3f00163c688374a9bfeb				
ePlus_dr[227:0]		104d3928ebf3c425c20364f3f59fe13a1c1ff667a6a3ff5e6091ff92b9			
edr_modN[227:0]		004d3928ebf3c425c20364f3f59fe1cfeef3d77689022351b35a3687c			
k_invPlus_edr[227:0]		0dd3e987ffc63e3e221d541c9c3b7cf89185eb7405c4c63d918c6d375			
multiN2_done					
dr[227:0]		08f1f3b406aeca3b3e8ab5d5a76a41165b54afda807ae85c3ab029d38			
multiN2_out[227:0]		0dd3e987ffc63e3e221d541c9c3b7cf89185eb7405c4c63d918c6d375			
start_gen					
kG_done					
hm[227:0]	075b4574e544f9ea8378af1e4e35a023c1aab69fe9c50d89ce6fcf581				
ecdsa_r[227:0]	03a9acc0899570fe8d3c142e646041f25d3a2a2bd1f80092e167e5c17				
ecdsa_s[227:0]	0dd3e987ffc63e3e221d541c9c3b7cf89185eb7405c4c63d918c6d375				
edr_modN[227:0]	004d3928ebf3c425c20364f3f59fe1cfeef3d77689022351b35a3687c				
emodN[227:0]	075b4574e544f9ea8378af1e4e35a023c1aab69fe9c50d89ce6fcf581				
pvk_d[223:0]	249ee54a927ae54a927aff8d24f5ff8d49eae5f93d53e474f572986				
rnd_k[223:0]	00				

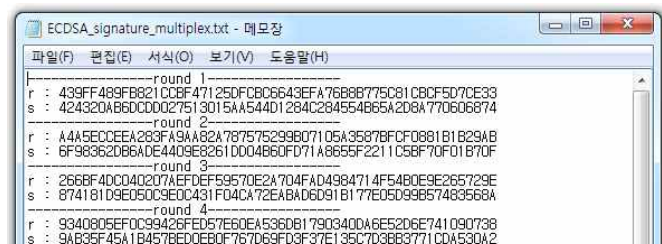
<그림 3-25> ECDSA 서명 하드웨어 시뮬레이션 결과

구현된 모듈의 높은 신뢰성을 얻기 위해서는 수많은 테스트벡터 값을 이용해 검증해야 한다. 따라서 서명 생성 모듈의 테스트벡터를 원하는 만큼 만들 수 있도록 하였으며, 서명 값 r, s 를 검증모듈에서 활용할 수 있도록

구현하였다.



<그림 3-26> ECDSA 서명 다중 생성



<그림 3-27> ECDSA 서명 다중 생성 결과

<그림 3-28>과 같이 서명 생성을 다중으로 생성하여 테스트벡터를 만들고 서명 검증 모듈에서 활용할 수 있다.



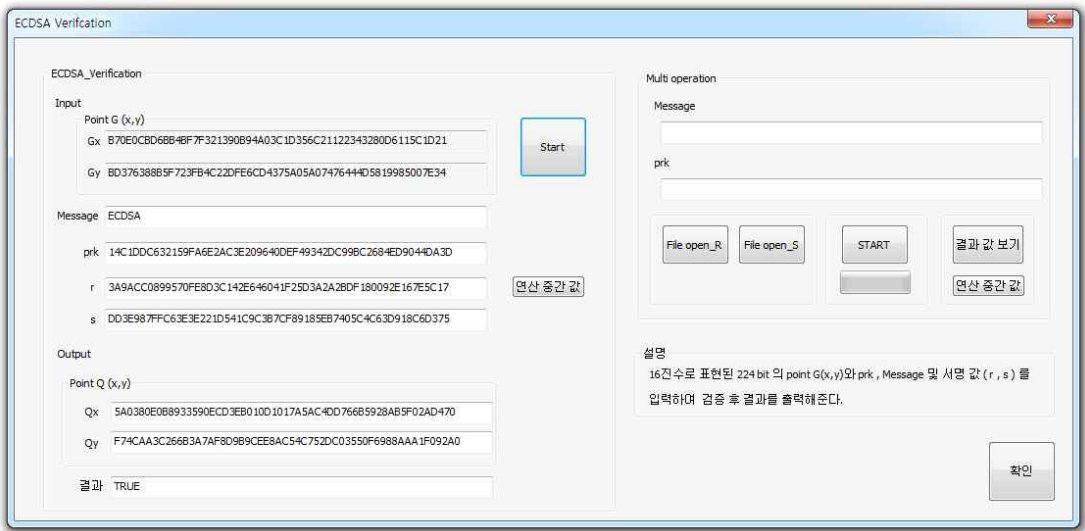
<그림 3-28> ECDSA 서명 다중 생성 중간 값

7. ECDSA 검증 모듈 검증

ECDSA 검증 모듈은 ECDSA 서명 값을 검증하기 위한 모듈로 앞서 설명한 모든 모듈을 이용해 구현된다. 검증 모듈을 구현할 때는 사전에 비밀 키를 공유했다고 가정한 후 검증을 해야 한다. ECDSA 검증 테스트 벡터와 검증 결과는 <표 3-20>과 같으며, 검증 모듈의 검증 소프트웨어 결과는 <그림 3-29>이다.

<표 3-20> ECDSA 검증 테스트벡터 예

$G(x, y)$	x : B70E0CBD6BB4BF7F321390B94A03C1D356C21122343280D6115C1D21
	y : BD376388B5F723FB4C22DFE6CD4375A05A07476444D5819985007E34
r	3A9ACC0899570FE8D3C142E646041F25D3A2A2BDF180092E167E5C17
s	DD3E987FFC63E3E221D541C9C3B7CF89185EB7405C4C63D918C6D375
d	14C1DDC632159FA6E2AC3E209640DEF49342DC99BC2684ED9044DA3D
m	ECDSA
결과	TRUE



<그림 3-29> ECDSA 검증모듈 검증

ECDSA 검증 모듈의 계산 순서대로 중간 값을 텍스트파일에 출력하므로

연산과정을 검증할 수 있다. 연산 과정은 다음과 같다.

해시함수를 이용해 메시지 해시 값 $e = \text{SHA1}(m)$ 를 구한다. 해시 값을 모듈러 N 연산을 하면 $e \bmod N$ 을 구할 수 있다. 다음으로 서명 값 s 를 역원 연산을 하여 w 를 구한다.

$$w = s^{-1} \bmod N \quad (3.13)$$

이어서 앞서 구한 e 와 w 를 곱하여

$$u_1 = ew \bmod N \quad (3.14)$$

u_1 을 구한다. u_2 는 서명 값 r 과 w 를 곱하여 구할 수 있다.

$$u_2 = rw \bmod N \quad (3.15)$$

다음으로 u_1G 와 u_2Q 는 스칼라 곱셈을 이용해서 구한다. 그리고 타원곡선 위의 두 점 덧셈 모듈을 이용하여 $u_1G + u_2Q$ 을 계산한다. 계산된 값 x, y 에서 x 값은 t 와 같으며 t 의 값이 r 값과 일치하면 true이다.



<그림 3-30> ECDSA 검증 중간 값

앞서 서명 모듈에서 다중 생성을 통해 만들어진 서명 값 r, s 를 검증하기 위해 검증 모듈 또한 다중으로 동작할 수 있도록 구현하였다.



<그림 3-31> ECDSA 검증모듈 다중 검증



<그림 3-32> ECDSA 검증모듈 다중 검증 중간 값

본 ECDSA 검증 소프트웨어를 이용하여 ECDSA 하드웨어 모듈을 검증하였다. 개발과정에서부터 하드웨어 모듈의 검증을 거치므로 손쉽게 내부 모듈 구현을 진행할 수 있었다. ECDSA의 내부모듈은 다양하며 개발과정에서

오류가 있을 때 어떤 모듈에서 오류가 있는지 찾기가 쉽지 않으나, 하드웨어로 암호 모듈을 개발할 때 소프트웨어를 같이 병행하여 개발하면 모듈의 기능 검증을 손쉽게 할 수 있으며 개발속도를 향상시킬 수 있다.

제4장 결론

현재 CMVP의 타원곡선 전자서명의 검증방식은 모듈 전체에 대한 입·출력 값을 검증하므로 검증이 실패하면 어떤 내부 모듈이 잘못되었는가에 대한 피드백은 없다. 하드웨어 암호 모듈을 구현하는 방법은 직렬구조, 병렬구조, 파이프라인 구조 등으로 다양하여 각 서브 모듈의 기능을 검증 할 수 있는 테스트 벡터를 제공하는 소프트웨어의 개발이 병행되어야 한다. 이러한 요구사항을 만족시키기 위해서는 타원곡선 암호의 구현에 필요한 내부 모듈의 검증 과정이 필요하다.

따라서 본 논문에서는 타원곡선 전자서명을 하위 모듈부터 상위모듈까지 전체적인 검증을 하기 위한 프로그램을 개발하였다. 본 소프트웨어의 신뢰성을 높이기 위해 기존에 검증된 소프트웨어를 이용해 테스트를 수행하였다. 본 프로그램의 기능은 타원곡선 전자서명의 하위 모듈인 유한체의 덧셈, 뺄셈, 곱셈, 역원 연산 모듈이 존재한다. 이러한 유한체 사칙연산 모듈을 이용하여 상위 모듈인 타원곡선 위의 점 연산 모듈을 구현할 수 있으며, 이 연산을 이용해 스칼라 곱셈이 가능하다. 또한, 상위 모듈인 타원곡선 전자서명의 서명 생성 및 서명 검증 모듈의 테스트벡터를 제공하고 있다.

본 프로그램은 전체 내부 모듈의 테스트벡터를 제공하기 때문에 암호 모듈 개발자가 개발 과정에서 모든 내부 모듈을 점검할 수 있다. 개발 과정에서 내부 모듈을 점검할 수 있으므로 개발의 진전속도 향상을 기대할 수 있다.

본 소프트웨어는 224비트의 ECDSA 내부 모듈의 테스트벡터를 제공하고 있으나 향후 256비트, 384비트와 같이 더 강력한 보안을 제공하는 ECDSA의 테스트벡터를 제공하는 소프트웨어를 구현할 계획이다. 또한, ECDSA 이외의 암호 모듈 하드웨어 개발 시 필요한 테스트벡터 제공 소프트웨어 개발을 할 계획이다.

참 고 문 헌

- [1] 경동욱. (2004). AES(Rijndael) 암호 시스템의 효율적인 하드웨어 모듈 설계 및 검증도구 설계. 석사학위 청구논문 : 부산대학교 대학원
- [2] 경동욱 · 김동규. (2004). 암호 하드웨어 모듈의 신뢰성 검증도구. 한국정보과학회 학술발표논문집, 31(1A), 265-267.
- [3] 김영자. (2004). 타원 곡선 암호 기반 보안 메신저 시스템의 설계 및 구현. 석사학위 청구논문 : 중앙대학교 대학원
- [4] 김요식. (2005). 암호 알고리즘 표준 적합성 검증도구 설계 및 구현. 석사학위 청구논문 : 공주대학교 대학원
- [5] 김정대 · 한성일 · 이강수. (2005). CMVP 검증을 위한 FIPS 140-2 기반 검증 업무 프로그램. 한국멀티미디어학회 학술발표논문집, 574-578.
- [6] 김태훈 · 정석원. ECDSA 하드웨어 테스트 벡터를 위한 SW 구현. 한국정보보호학회 동계학술대회 논문집, 1-4
- [7] 김태훈 · 홍휘승 · 김성훈 · 김현곤 · 이경효 · 정석원. 224비트 유한체 곱셈 하드웨어 구조 검증을 위한 소프트웨어 설계 및 구현. 한국정보보호학회 하계학술대회 논문집, 23(1), 139-142
- [8] 목포대학교 산학협력단. (2013.05.23). 몽고메리 역원 알고리즘을 위한 뺄셈 연산 장치 및 그 방법. 특허등록(10-1321259).
- [9] 박영호. (2007). 공개키 암호. 한국물리학회 특집호, 21(2), 7-12.
- [10] 서창호 · 홍도원 · 윤보현 · 김석우 · 이옥연 · 정교일. (2004). 정보보안: 무선 환경에 적합한 타원곡선 암호 알고리즘의 검증도구. 정보처리학회논문지 C, 11(5), 569-576.
- [11] 신현구. (2005). 표준 스트림 암호 구현 적합성 검증 방법. 석사학위 청구논문 : 국민대학교 대학원

- [12] 양인제 · 경동욱 · 김동규. (2003). 암호 하드웨어 모듈의 테스트를 위한 검증 도구. 한국멀티미디어학회 학술발표논문집, 267-270.
- [13] 윤은준. (2012). ElGamal 암호 알고리즘 기반 이메일 전송 프로토콜. 대한전자공학회 학술대회, 1950-1952.
- [14] 이상진 · 김창한 · 장남수 · 윤택영. (2006). 인수분해 전용 하드웨어 연구 동향. 정보보호학회지, 16(4), 7-14.
- [15] 이영석. (2013). 암호 프로그램 검증 시스템 설계 및 구현. 한국정보통신학회 논문지, 17(4), 869-877.
- [16] 정창호. (2004). 암호 모듈 구현 적합성 검증을 위한 웹 서버 기반 시스템. 석사학위 청구논문 : 고려대학교 정보보호대학원
- [17] 차재원. (2004). 웹카메라 서버용 시스템에서 ECDSA를 이용한 상호인증모듈 구현에 관한 연구. 석사학위 청구논문 : 전남대학교 대학원
- [18] 하경주 · 서창호 · 김대엽. (2014). 암호 알고리즘 구현 적합성 평가 시스템 설계. 한국통신학회논문지, 39(4), 242-250.
- [19] Barker, E., Barker, W., Burr, W., Polk, W., & Smid, M. (2006). Recommendation for key management-part 1: General (revised). In NIST special publication.
- [20] FIPS, P. (2001). 140-2: Security requirements for cryptographic modules. National Institute of Standards and Technology.
- [21] Hall, T. A., & Keller, S. S. (2014). The FIPS 186-4 Elliptic Curve Digital Signature Algorithm Validation System (ECDSA2VS).
- [22] Hankerson, D., Vanstone, S., & Menezes, A. J. (2004). Guide to elliptic curve cryptography. Springer.
- [23] Johnson, D., Menezes, A., & Vanstone, S. (2001). The elliptic curve digital signature algorithm (ECDSA). International Journal of Information Security, 1(1), 36-63.

- [24] National Institute of Standards and Technology Communications Security Establishment Canada. (2014). Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program.
- [25] Polk, W. T., Dodson, D. F., & Burr, W. E. (2005). Cryptographic algorithms and key sizes for personal identity verification. US Department of Commerce, Technology Administration, National Institute of Standards and Technology.
- [26] PUB, F. (2012). Secure Hash Standard (SHS).
- [27] SEC, S. (2010). 2: Recommended elliptic curve domain parameters. See <http://www.secg.org>.
- [28] Yoon, J. C., Jung, S. W., & Lee, S. (2004). Architecture for an elliptic curve scalar multiplication resistant to some side-channel attacks. In Information Security and Cryptology-ICISC 2003 139~151. Springer Berlin Heidelberg.

224비트 ECDSA 하드웨어 기능 검증을 위한 소프트웨어 구현

김 태 훈

목포대학교 대학원 정보보호기술학협동과정
(지도교수 정 석 원)

〈국문초록〉

현재 CMVP에서 타원곡선 전자서명의 검증방식은 모듈 전체에 대한 입·출력 값을 검증하므로 검증이 실패하면 어떤 내부 모듈이 잘못되었는가에 대한 피드백은 없다. 또한 하드웨어 암호 모듈을 구현하는 방법은 직렬구조, 병렬구조, 파이프라인 구조 등으로 다양하여 각 구조에 필요한 테스트 벡터가 다르며 이에 대한 테스트 벡터를 제공하는 소프트웨어의 개발이 병행되어야 한다. 이러한 요구사항을 만족시키기 위해서는 타원곡선 암호의 구현에 필요한 내부 모듈의 검증 과정이 필요하다. 이를 위해 본 논문에서는 타원곡선 전자서명을 하위 모듈부터 상위모듈까지 전체적인 검증을 하기 위한 프로그램을 개발하였다. 본 프로그램은 전체 내부 모듈의 테스트벡터를 제공하기 때문에 암호 모듈 개발자가 개발 과정에서 모든 내부 모듈을 점검할 수 있다. 개발 과정에서 내부 모듈을 점검할 수 있으므로 개발의 진전속도 향상을 기대할 수 있다. 향후 ECDSA 이외의 암호 모듈 하드웨어 개발 시 필요한 테스트벡터 제공 소프트웨어 개발을 할 계획이다.

핵심용어 : ECDSA, CMVP, 하드웨어 검증, 검증 소프트웨어