情報保護學會論文誌
第 22 卷 第 5 號, 2012. 10

임베디드 장비 상에서의 공개키 기반 암호를 위한 다중 곱셈기 최신 연구 동향*

서 화 정,† 김 호 원‡
부산대학교

Research on Multi-precision Multiplication for Public Key Cryptography over Embedded Devices*

Hwajeong Seo,† Howon Kim‡
Pusan National University

요 약

공개키 기반 암호화 상에서의 다중 곱셈 연산은 높은 복잡도로 인해 성능 개선을 위해서는 우선적으로 고려되어야 한다. 특히 임베디드 장비는 기존의 환경과는 달리 한정적인 계산 능력과 저장 공간으로 인해 높은 복잡도를 나타내는 공개키 기반의 암호화를 수행하기에는 부적합한 특성을 가진다. 이를 극복하기 위해 다중 곱셈 연산을 빠르게 연산하고 적은 저장공간을 요구하는 기법이 활발히 연구되고 있다. 본 논문에서는 자원 한정적인 센서 네트워크 상에서의 효율적인 공개키 기반 암호화 구현을 위한 다중 곱셈기의 최신 연구 동향을 살펴본다. 이는 앞으로의 센서 네트워크상에서의 공개키 기반 암호화 구현을 위한 참고자료로서 활용이 가능하다.

ABSTRACT

Multi-precision multiplication over public key cryptography should be considered for performance enhancement due to its computational complexity. Particularly, embedded device is not suitable to execute high complex computation, public key cryptography, because of its limited computational power and capacity. To overcome this flaw, research on multi-precision multiplication with fast computation and small capacity is actively being conducted. In the paper, we explore the cutting-edge technology of multi-precision multiplication for efficient implementation of public key cryptography over sensor network. This survey report will be used for further research on implementation of public key cryptography over sensor network.

Keywords: Public Key Cryptography, Sensor Network, Embedded Device, Multi-precision Multiplication

I. 서 론

대표적인 공개키 기반 암호화인 RSA와 ECC 그리고 Pairing 연산은 암호화 과정을 수행하기 위해 다중 곱셈 연산을 유한체 상에서 여러 번 수행해야 한다. 해당 유한체 상에서의 다중 곱셈 연산은 공개키 기반 암호화에서 전체 연산 부하의 70% 이상을 차지하기 때문에 연산을 보다 효율적으로 구현하는 것은 무엇보다 중요하다. 지금까지는 공개키 기반 암호화의 높은 복잡도로 인해 기존의 개인 컴퓨터 상에서의 공개키 기반 암호화의 구현만이 현실적으로 가능하였다. 따라서 임베디드 장비 상에서는 대칭키 기반 암호화인 AES를 통해서 메시지의 암복호화가 수행되었다. 그

접수일(2012년 8월 14일), 게재확정일(2012년 9월 12일)
* 이 논문은 2010년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No.2010-0026621).
† 주저자, hwajeong@pusan.ac.kr
‡ 교신저자, howonkim@pusan.ac.kr

렇지만 대칭키 암호화는 키 관리에 문제점을 가지고 있을 뿐 아니라 공개키 기반의 암호화에 비해 그 가능한 서비스의 폭이 좁다. 따라서 실용적인 응용을 위해서는 공개키 기반 암호화의 구현이 요구되어 진다. 이에 대한 해결책으로 최근에는 임베디드 장비 상에서의 공개키 기반 암호화를 실현하기 위해 다양한 곱셈기법에 대한 연구가 진행되고 있다. 이는 유비쿼터스 사회에 진입함에 따라 임베디드 장비를 통한 다양한 서비스(홈네트워크, 스마트그리드, e-헬스)의 실현이 가능해지고 앞으로 더 많은 서비스의 창출이 기대되기 때문이다. 따라서 현재 자원 한정적인 임베디드 장비 상에서의 빠르고 적은 용량을 요구하는 다중 곱셈 기법이 보다 활발히 연구되고 있다. 특히 임베디드 장비상에서의 구현 시에는 레지스터 상에서의 연산에 비해 많은 clock cycle을 소모하는 메모리 연산을 보다 효율적으로 줄이는 것이 성능을 가속화시킬 수 있는 기법이다. 따라서 임베디드 장비 상에서의 메모리 접근을 줄여 곱셈 연산에 소모되는 부하를 줄이는 것이 소프트웨어를 통한 공개키 기반 암호화의 구현에 가장 중요하다. 본 논문에서는 임베디드 장비 상에서의 최신 동향에 대해 알아본다. 해당 연구 결과는 최근 프라임과 바이너리 체상에서의 곱셈을 모두 포괄한다. 따라서 본 논문에서 소개되는 여러 기법들이 추후 많은 연구자들의 연구에 도움이 될 수 있기를 바란다. 본 논문의 구성은 다음과 같다. 2장에서는 임베디드 장비 상에서의 연산과 주소기법 그리고 하드웨어 기반 곱셈기에 대해 설명한다. 3장에서는 Prime Field상에서의 곱셈기 법에 대해 알아본다. 4장에서는 Binary Field 상에서의 곱셈기법을 알아본다. 마지막으로 5장에서는 본 논문에 대한 결론을 내리고 해당 논문을 마무리 짓는다.

II. 임베디드 장비 상에서의 연산

본 장에서는 8, 16-bit 임베디드 장비 상에서의 연산에 대해 살펴보며 이를 통해 장비에 따른 연산의 효율성을 확인해 보는데 도움이 되도록 한다.

2.1 ATmega128 8-bit processor

ATmega128은 8-bit processor로서 7.3728Mhz로 동작한다. 프로그램은 128KB EEPROM에 저장되며 데이터 메모리를 위해 4KB RAM을 제공한다[10,11]. ATmega128은 전형적인 RISC 구조로서 32개의 레지스터를 가진다. 다양한 instruction set을 제공하여 원하는 연산을 효율적으로 구성하는 것이 가능하다. [표 1]은 ATmega128에서 사용가능한 Instruction set을 나타낸다. 인자에 대한 저장과 불러오기 연산의 경우 메모리에 대한 접근을 시도해야 하기 때문에 2 clock cycle이 소모된다. 레지스터에 대한 연산의 경우 보다 효율적인 연산이 가능함으로 1 clock cycle에 연산이 수행된다.

2.2 MSP430 16-bit processor

MSP430은 16-bit processor로서 8MHz로 동작하며 48KB의 program flash memory와 10KB의 RAM을 제공한다. 총 12개의 general purpose register를 제공하며 하드웨어 기반의 곱셈연산이 가능하여 prime field 상에서의 곱셈에 유리하다[12,13].

[표 2]는 MSP430 상에서의 연산을 나타낸다. ATmega128에서와 동일하게 메모리에 대한 접근 시에 보다 많은 부하가 소모된다. 따라서 레지스터를 통

[표 1] ATmega128의 Instruction set

방식	표기	동작	#clock
Load from Y	$ld\ R_d, Y$	$R_d \leftarrow (Y)$	2
	$ld\ R_d, Y+$	$R_d \leftarrow (Y), Y \leftarrow Y+1$	2
	$ld\ R_d, Y+q$	$R_d \leftarrow (Y+q)$	2
Store to X	$st\ X, R_s$	$(X) \leftarrow R_s$	2
	$st\ X+, R_s$	$(X) \leftarrow R_s, X \leftarrow X+1$	2
Copy Register	$mov\ R_d, R_s$	$R_d \leftarrow R_s$	1
Addition of the register	$add\ R_d, R_s$	$R_d \leftarrow R_s + R_d$	1

R_d : 목적지 레지스터
R_s : 근원지 레지스터

〔표 2〕 MSP430의 instruction set

방식	표기	동작	#clock
Load to X	$mov\ R_s, X(R_d)$	$(X+R_d) \leftarrow R_s$	4
	$mov\ @R_s, X(R_d)$	$(X+R_d) \leftarrow (R_s)$	5
Load to label	$mov\ X(R_s), \&label$	$(label) \leftarrow (X+R_s)$	6
	$mov\ R_s, \&label$	$(label) \leftarrow R_s$	4
	$mov\ @R_s+, \&label$	$(label) \leftarrow (R_s)$ $R_s \leftarrow R_s+2$	5
	$mov\ @R_s, \&label$	$(label) \leftarrow (R_s)$	5
Load to Register	$mov\ X(R_s), R_d$	$R_d \leftarrow (X+R_s)$	3
Copy Register	$mov\ R_s, R_d$	$R_d \leftarrow R_s$	1
Clear Register	$clr\ R_d$	$R_d \leftarrow R_d \oplus R_d$	1
Addition of the register	$add\ R_s, R_d$	$R_d \leftarrow R_s + R_d$	1
	$addc\ R_s, R_d$	$R_d \leftarrow R_s + R_d + C$	1
	$adc\ R_d$	$R_d \leftarrow R_d + C$	1

R_d: 목적지 레지스터, R_s: 근원지 레지스터, X,$label$: 간접 주소방식, C:Carry bit

한 연산의 구현이 요구되어 진다.

메모리에 대한 접근 방식에서는 ATmega128에서는 모든 접근 방식이 동일하게 2 clock cycle안에 실행되었다면 MSP430에서는 사용하는 메모리 접근 방식에 따라 요구되는 clock cycle이 달라진다. 이에 대해서는 다음 장에서 자세히 설명하도록 한다.

2.3 메모리 접근방식

MSP430의 경우 메모리 접근 방식에 따라 소모되는 clock cycle이 달라진다. 따라서 register에 주소값을 미리 불러와서 접근하는 indirect 방식이나 register에서 값을 바로 불러오게 될 경우에 적은 clock cycle을 소모하며 동작을 수행할 수 있는 장점을 가진다. 〔표 3〕은 두 마이크로 프로세서에 동일하게 존재하는 주소 접근 방식을 나타낸다. 가장 많이 사용되는 방식은 간접 주소방식이며 Increment 주소 방식은 간접주소 접근 이후에 offset을 자동으로 조정해 주는 기법이다. 따라서 보다 효율적으로 주소 접근이 가능하다.

2.4 곱셈기법

본 장에서는 대표적인 마이크로 프로세서인 ATmega128과 MSP430 상에서의 효율적인 곱셈기법을 살펴본다. 두 마이크로 프로세서는 Word 크기와 General Purpose Register의 개수 그리고 Instruction set의 특성에서 다르다. 따라서 구현에 사용되는 기법에서도 차이가 발생할 뿐 아니라 결정적으로 하드웨어 형태로 제공되는 곱셈기에서 차이가 발생한다. ATmega128은 하드웨어 곱셈기를 제공하여 효율적인 8-bit 곱셈연산이 가능하다. 해당 연산은 가장 기본적인 곱셈연산으로서 간단하지만 빠른 성능을 위해서는 보다 복잡한 구조가 필요하다. 이에 따라 MSP430에서는 연산의 결과를 축적해서 계산이 가능한 MAC기법을 가능하게 하였다. 해당 연산은 곱셈되는 결과 값을 하드웨어의 메모리에 저장하고 업데이트를 함으로서 결과 값을 마지막 연산이 끝나는 시점에서 접근하면 되도록 설계되어 있다.

2.5 Mul

ATmega128 센서 모트의 경우에는 곱셈을 위한 instruction으로서 Mul 연산을 제공한다. 해당 연산은 두 개의 8-bit 의 레지스터에서 값을 불러온 후 곱셈을 수행하고 이에 대한 결과값을 이전에 인자를 불러올 때 사용했던 두 개의 8-bit 레지스터에 저장하게 된다. 따라서 2 clock cycle 안에 8-bit 곱셈이 가능한 장점을 가진다.

2.6 Mul and Acc (MAC)

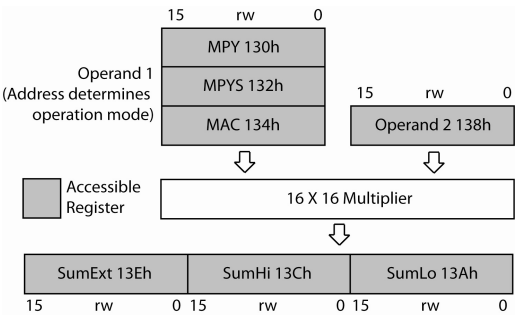
MSP430의 경우 보다 진보된 형태의 곱셈기를 제공한다. 기존의 곱셈기가 곱셈 결과를 바로 레지스터

〔표 3〕 주소 접근 방식

방식	표기	동작
Direct	R_d	직접 메모리에 접근해서 값을 가지고 온다.
Indexed	$X(R_d)$	offset과 간접 접근 방식을 통해 메모리에 접근한다.
Indirect	$@R_d$	간접 접근 방식으로 메모리에 접근한다.
Increment	$@R_d+$	간접 접근 방식 후 자동으로 offset을 갱신해준다.

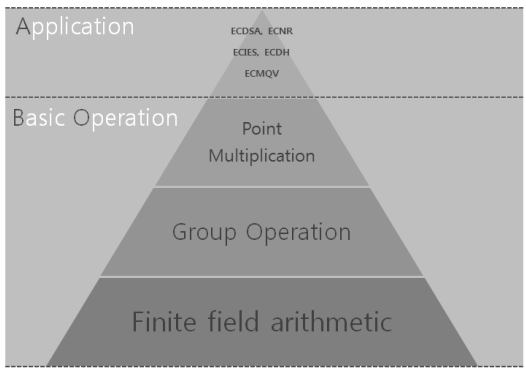
에 돌려주는 형식을 취했다면 해당 곱셈은 결과값을 지속적으로 하드웨어 곱셈기 내에 위치한 메모리에 축적 후 마지막 곱셈 연산이 끝나는 시점에 결과값에 접근하여 값을 가져오게 된다. 따라서 중간 결과값에 대한 접근이 불필요하게 되므로 메모리에 대한 접근을 효율적으로 줄일 수 있다. 〔그림 1〕은 하드웨어 곱셈기의 block diagram으로서 회색으로 칠해진 부분이 곱셈에 사용되기 위한 입력과 출력에 해당하는 부분이다. 먼저 MPY, MPYS, MAC 중에서 사용하고자 하는 곱셈 기법에 해당하는 메모리 주소에 첫 번째 곱셈인자를 인가시킨다. 이를 통해 곱셈을 수행하기 위한 준비가 된다. 그리고 두 번째 인자를 Operand2에 인가시키게 되면 곱셈연산이 수행되면서 해당 결과값을 SumLo, SumHi, SumExt에 차례적으로 돌려주게 된다. 여기서 SumExt는 지속적으로 값을 축적시키는 MAC연산의 경우 발생하는 올림값을 발생시켜 주게 된다.

대표적인 공개키 기반 암호화인 RSA, ECC 그리고 Pairing의 경우 암호화 과정을 수행하기 위해 다중 곱셈 연산을 체 상에서 여러 번 수행해야 한다. 특히 〔그림 2〕에서와 같이 타원곡선 상에서의 다중 곱셈 연산은 가장 근본적인 연산으로서 전체 타원 곡선 연산의 70%를 차지한다. 해당 유한체 연산은 상위 단에 위치한 그룹연산과 스칼라 곱셈을 수행하기 위해서 필요하며 또한 이 연산들은 ECDSA, ECDH를 수행하기 사용된다. 따라서 효율적인 다중 곱셈 연산의 구현은 공개키 기반 암호화 구현에 가장 중요한 요소로 작용한다. 또한 자원 한정적인 임베디드 장비 상에서의 곱셈 연산의 구현은 메모리에 대한 접근을 보다 효율적으로 줄이는 것이 성능으로 직결되어 나타난다. 이를 통해 생각해 볼 때 임베디드 장비 상에서의 메모리 접근을 보다 효율적으로 줄여 곱셈 연산에 소모되는 부하를 줄이는 것이 공개키 기반 암호화의 구현에 가장 중요하다.



〔그림 1〕 MSP430 하드웨어 곱셈기

본 장에서는 곱셈을 보다 이해하기 쉽도록 마름모 형식으로 나타낸다. 마름모의 꼭지점은 곱셈을 나타내며 화살표는 곱셈방향 그리고 세로방향은 결과값의 주소를 나타낸다. 또한 왼쪽 그림은 전체 곱셈이 수행되는 순서와 위치를 나타낸 블락도이다.

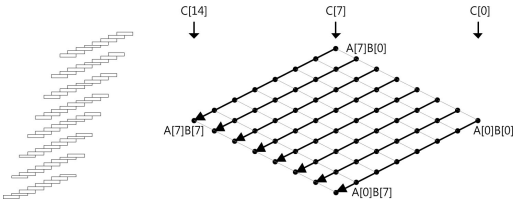


〔그림 2〕 타원곡선 암호화 전체 구성도

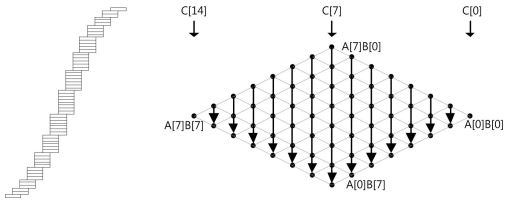
2.7 School Book 곱셈기

학교에서 배우는 곱셈 방식을 그대로 임베디드 장비에 적용하는 방식이 초창기에는 많이 적용되었다.

Row-wise 곱셈기는 같은 행에 위치한 모든 인자값을 한 번에 읽어와서 곱셈을 취하는 방식이다. 해당 방식은 하나의 인자값에 대한 모든 곱셈 결과를 얻을 수 있다는 장점이 있지만 지속적으로 중간 결과값과 인자값을 불러와야 하는 단점을 가진다.



〔그림 3〕 Row-wise 곱셈기



〔그림 4〕 Column-wise 곱셈기

Row-wise 곱셈기에 대한 예시는 〔그림 3〕에 자세히 나타나 있다. 〔그림 3〕에서는 총 8개의 A, B인자에 대한 곱셈을 통해 15개의 C결과값을 도출해 내는 연산을 수행한다. A〔0〕이 인자로 불러오게 되면 B의 경우 0~7에 해당하는 모든 인자값을 불러와 A〔0〕과의 부분 곱셈값을 도출하게 된다. 해당 결과값은 C의 0~8에 해당하는 메모리 주소에 값을 저장하게 된다. 그 다음 연산은 A〔1〕에 대한 부분 곱셈을 수행하게 되며 지속적으로 모든 A의 인자에 대한 곱셈 연산을 수행하게 된다. 따라서 B에 대한 인자를 모두 저장하기 위한 레지스터와 결과값을 저장하기 위한 레지스터를 모두 확보해야하는 구조를 가진다. 만약 충분한 레지스터가 보장이 된다면 매우 효과적으로 연산이 가능하지만 임베디드 장비는 한정적인 레지스터로 인해 효율적인 연산을 기대하기 힘들다. 따라서 Row-wise 곱셈기는 임베디드 장비 상에서는 효율적인 곱셈 기법이 아니다.

Column-wise 곱셈기는 같은 열에 위치한 부분 곱셈을 한 번에 모두 수행하는 방식으로서 중간 결과값을 다시 읽어오지 않아도 되는 장점을 가진다. 해당 기법은 동일한 열에 위치한 인자들을 다 읽어와 연산을 수행하고 해당 열의 모든 부분 곱셈을 수행하고 난 다음에는 다시 돌아와서 해당 열을 계산할 필요가 없으므로 지속적인 메모리 접근을 효율적으로 줄일 수 있다.

〔그림 4〕에서 나타난 Column-wise 곱셈기 기법에서는 화살표 방향을 보면 동일한 결과값 C의 인자에 대한 부분 곱셈을 한 번에 다 수행함을 확인할 수 있다. 이는 인자를 불러오기 위한 연산의 증가를 불러오지만 축적해야하는 레지스터의 크기가 곱셈결과값보다 하나가 크기 때문에 레지스터의 수가 적은 임베디드 장비상에서는 보다 효율적으로 연산이 가능하다. C〔7〕열을 예시로 들어보면 A〔7〕과 B〔0〕 그리고 A〔6〕과 B〔1〕과 같이 동일한 열에 대한 연산이 수행되므로 결과값의 축적이 매우 용이하다.

Karatsuba 곱셈기는 연산 부하가 큰 곱셈 연산을 덧셈과 뺄셈같은 부하가 작은 연산으로 대치하여 보다 효율적으로 연산이 가능하도록 한다. 하지만 덧셈과 뺄셈으로 대치하여 연산한 중간 과정값을 모두 레지스터에 유지하고 있어야 한다는 단점으로 인해 자원한정적인 임베디드 장비에는 비효율적이다〔6〕. 그 이유는 임베디드 장비의 경우 사용가능한 레지스터의 수가 한정적이기 때문이다.

다음 수식은 Karatsuba 곱셈이 연산되는 원리를 나타내고 있다. 먼저 인자 x와 y의 경우 B를 기저로 가지도록 하며 동일한 크기를 가졌다고 가정한다.

$$x = x_1 B^m + x_0, \quad y = y_1 B^m + y_0 \tag{1}$$

해당 인자에 대한 곱셈은 다음 수식 (2)와 같이 나타낼 수 있다.

$$xy = (x_1 B^m + x_0)(y_1 B^m + y_0) = z_2 B^{2m} + z_1 B^m + z_0 \tag{2}$$

여기서 z_2, z_1, z_0은 수식 (3,4,5)와 같이 정의된다.

$$z_2 = x_1 y_1 \tag{3}$$

$$z_1 = x_1 y_0 + x_0 y_1 \tag{4}$$

$$z_0 = x_0 y_0 \tag{5}$$

수식 (2)연산의 경우 4번의 곱셈연산이 수행됨을 확인할 수 있다. 이를 Karatsuba연산으로 수행하게 되면 보다 적은 곱셈연산을 통해서 연산이 가능하다. 수식 (6,7)은 동일하게 곱셈연산을 수행하지만 수식 (8)의 경우에는 곱셈연산이 한번만 수행됨을 확인할 수 있다.

$$z_2 = x_1 y_1 \tag{6}$$

$$z_0 = x_0 y_0 \tag{7}$$

$$z_1 = (x_1 + x_0)(y_1 + y_0) - z_2 - z_0 \tag{8}$$

수식 (8)은 수식 (9)와 같이 풀어서 나타내면 동일한 연산을 수행하고 있음을 확인할 수 있다.

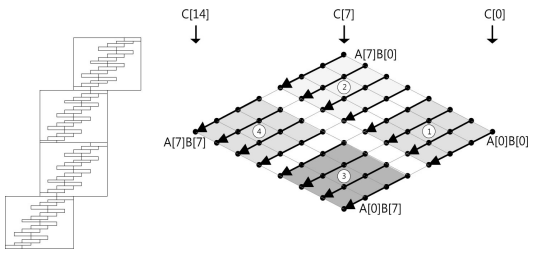
$$z_1 = x_1y_0 + x_0y_1 \quad (9)$$
$$= (x_1y_1 + x_1y_0 + x_0y_1 + x_0y_0) - x_1y_1 - x_0y_0$$
$$= (x_1 + x_0)(y_1 + y_0) - x_1y_1 - x_0y_0$$

III. Prime Field

3.1 Hybrid 곱셈기(2004년)[5]

[그림 5]에 나타난 Hybrid 곱셈기는 기존 Row-wise 곱셈기와 Column-wise 곱셈기 각각의 장점인 한 번에 곱셈에 필요한 모든 인자들을 불러와서 곱셈하는 기법과 결과값을 축적하는 레지스터의 개수를 최소화하는 방안을 혼합하여 사용하는 기법이다. 전체 곱셈은 큰 블락 단위로 묶어서 나타낼 수 있다. 해당 블락은 Column-wise형식으로 연산이 수행되며 블락은 다시 여러번의 Row-wise곱셈이 내부에서 수행되는 방식으로 설계되어 있다 해당 기법은 레지스터에 대한 효율성을 극대화함으로서 메모리에 대한 접근을 줄이게 된다.

Hybrid 곱셈에 대한 예시는 [그림 5]에 나타나 있다. 큰 블락은 1, 2, 3 그리고 4와 같은 형식으로 총 16번의 부분곱셈을 수행하며 이는 하나의 블락으로 묶어서 표현한다. 곱셈은 번호 순으로 1, 2, 3 그리고 4의 차례로 수행되며 해당 블락의 내부에서는 Row-wise 곱셈방식으로 곱셈이 수행된다.



[그림 5] Hybrid 곱셈기

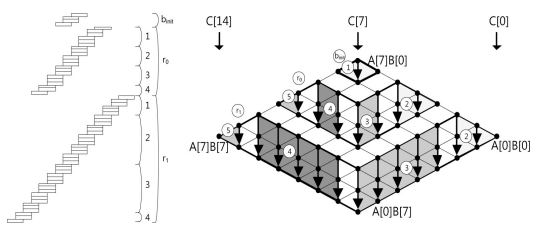
3.2 Column+MAC 곱셈기(2009년)[7]

해당 기법은 Mul and Acc 모드라는 특정한 instruction set에 한정된 기법이다. 기존의 hybrid 기법은 Mul만을 이용해서 구현하는 가장 효율적인 방법이었다면 해당 기법은 최신 임베디드 장비와 school book 기법인 Column-wise기법을 사용하여 최상의 성능이 나오도록 했다. 해당 기법은 MSP430에서 제공하는 MAC곱셈기 상에서 효율적이다. 기존의 곱셈기에서는 Mul을 하고난 결과값이 특정 레지스터에 바로 저장이 되는 구조로 설계되어 있었다. 따라서 해당 결과값을 바로 갱신해 주지 않으면 다음 결과값이 이전 결과값을 덮어 씌어버리는 문제점이 발생하여 온전한 결과값을 얻는 것이 불가능하였다. 하지만 MAC곱셈기의 경우에는 내부적으로 곱셈 결과값을 저장하는 메모리를 내장하고 있어 해당 결과값을 계산이 끝나는 시점에서 바로 갱신을 해주지 않더라도 마지막에 한 번만 저장을 해주게 되면 되는 효율적인 구조를 가진다.

3.3 Operand Caching 곱셈기(2011년)[1]

[그림 6]에 나타난 Operand Caching 기법은 기존의 기법들이 최대한 중간 결과값에 대한 접근을 피했다면 해당 기법은 중간 결과값에 대한 접근을 인자들에 대한 접근과 효율적으로 대치시키며 성능을 향상시켰다. 만약 곱셈에 수행되어야 하는 인자가 각각 10개씩 존재한다면 부분 곱셈을 10개의 쌍에 대해 취한 결과값을 구하는 것이 가능하다. 그 다음에 결과값을 구하기 위해 인자를 선택해야 하는 시점이 오면 이전에 사용했던 인자의 쌍 중 하나의 인자들의 집합을 다음 연산 결과에도 사용하여 인자에 대한 접근을 효율적으로 줄일 수 있는 구조로 되어 있다.

Operand Caching에 대한 예시는 [그림 6]과 같다. 먼저 가장 상위단에 위치하는 b_{init}을 계산한 후 다음 열에 해당하는 r_1을 계산한다. 이때 그림에서와 같이 2, 3, 4 그리고 5의 순서로 부분곱셈이 수행되게 된다. 여기서 2와 3은 A인자들의 집합을 공유하게 되며 3, 4 그리고 5의 연산에서는 B인자들의 집합을 공유하게 된다. 따라서 지속적으로 동일한 인자를 유지하며 사용하게 되므로 인자를 불러오기 위한 메모리



[그림 6] Operand Caching 곱셈기

〔표 4〕 ATmega128 상에서 160-bit 곱셈 연산

Method	Clock Cycle
Hybrid〔5〕	3,106
Operand Caching〔1〕	2,395

접근을 효율적으로 줄일 수 있다.

〔표 1,2〕는 160-bit 곱셈 연산을 대표적인 8-bit, 16-bit 센서 모트인 ATmega128과 MSP430상에서의 구현 결과를 나타낸다. 먼저 〔표 1〕에서는 Hybrid기법과 Operand Caching기법을 사용하여 곱셈을 구현하였을 경우를 비교하여 나타내고 있다. 기존의 Hybrid 방식에 비해 Operand Caching 기법은 한번에 2clock cycle이 소모되는 메모리에 대한 접근을 효율적으로 줄였다. 그 결과 성능이 대폭 향상되었음을 확인할 수 있다.

반면에 MSP430에서는 Operand Caching기법이 ATmega128에서만큼 높은 성능을 나타내지 못한다. 그 이유는 사용가능한 레지스터의 개수가 총 12개로 32개인 ATmega128에 비해 적어 인자들을 저장하기 위한 공간이 부족할 뿐 아니라 MAC곱셈기법이 제공됨에 따라 결과를 저장하기 위해 메모리에 접근이 많은 Operand Caching기법은 비효율적인 구조를 가진다. 그 결과 Column+MAC기법을 사용할 경우 이전에 알려진 구현에 비해 10%정도 향상된 결과값을 나타냄을 확인할 수 있다. 따라서 임베디드 장비의 독특한 구조와 특성 그리고 제공되는 기능에 따라 적합한 구조의 곱셈기법을 사용하는 것이 요구되어 지며 이는 앞으로 계속적인 발전으로 인해 새로운 구조가 제안됨에 따라 보다 가속화되리라 생각된다.

IV. Binary Field

4.1 Hybrid곱셈기 (2008년)〔8〕

한국에서 개발한 TinyECCK에서는 window크기만큼의 값을 사전에 계산하여 해당 결과값을 불러와서 바로 Xor연산을 수행하는 형식으로 계산이 수행된다. 해당 곱셈기법은 ATmega128 상에서 C언어를 통해 구현되었으며 놀랍게도 이전에 구현되었던 어셈블리 기반의 구현에 비해 성능이 향상됨을 확인 할 수 있었다. 이에 대한 구현은 4.3에서 최근에 소개된 기법과 비슷한 구조를 가지므로 4.3에서 자세한 설명을 하도록 한다.

〔표 5〕 MSP430 상에서 160-bit 곱셈연산

Method	Clock Cycle
Hybrid〔9〕	1,746
Column+MAC〔7〕	1,586

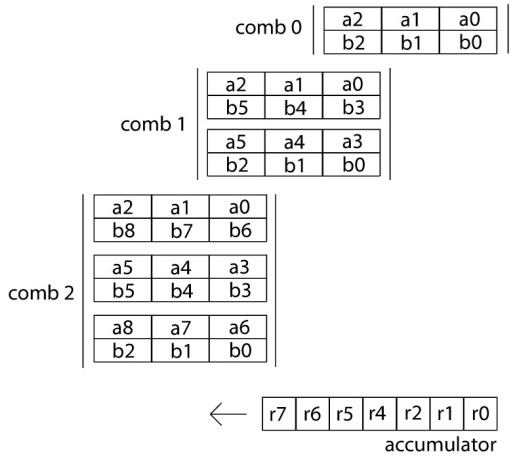
4.2 Block Comb 곱셈기 (2009년)〔2〕

한번 읽어온 인자에 대한 곱셈을 블락단위로 구성하여 곱셈 수행 시 지속적인 인자에 대한 접근이 일어나지 않도록 하였다. 기존의 Hybrid 곱셈기와 매우 유사한 기법이지만 기존의 Hybrid 기법은 Row 곱셈의 특성을 지녔다면 해당 기법은 Column 곱셈의 특성을 블락단위로 수행하여 성능을 향상 시켰다.

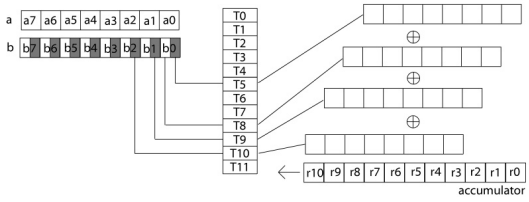
〔그림 7〕은 이에 대한 설명을 나타낸다. 먼저 인자들을 각각 3개씩 블락으로 묶어주며 이에 대한 곱셈을 수행한다. 예를들어 comb 0의 경우에는 b0의 첫 번째 bit를 확인하여 1인 경우 a0, a1 그리고 a2의 값을 결과값에 xor해준다. 이와 더불어 b1 그리고 b2도 첫 번째 인자의 값을 확인하여 결과값을 목적지 레지스터에 xor해준다. xor가 끝나고 나면 인자들을 1-bit씩 shift시켜주고 다음 bit에 대해 확인하여 이전 수행과정을 반복하게 된다.

4.3 Comb-Window 곱셈기 2011년〔3,4〕

곱셈을 연산 수행단계에 하지 않고 미리 계산해 놓은 연산결과에 접근하여 해당 값을 할당하는 방식으로 구현한 방법으로서 복잡한 곱셈연산 대신 메모리에 결과값을 저장해 놓은 이후 불러오는 형식으로 연산을 수행하여 효율성을 증진시켰다. 즉 사전에 하나의 인



〔그림 7〕 Block Comb 곱셈기



〔그림 8〕 Comb-Window 곱셈기

자 집합에 대한 모든 곱셈 결과값을 하나의 테이블 형식으로 계산하여 저장하게 된다. 해당 결과값은 다른 인자의 값을 주소값으로 사용하여 순차적으로 접근하여 결과값을 결과 레지스터에 xor하는 형식으로 계산한다.

〔그림 8〕에는 이에 대한 예시가 나와 있다. 사전에 인자 a에 대한 모든 곱셈 결과값을 계산한다. 계산된 결과값은 b인자의 값을 주소로 하여 접근하게 된다. 여기서는 총 2^4 개수의 a의 크기에 해당하는 결과값을 사전에 생성하여 저장하게 된다. 따라서 b인자에 의해 계산된 주소로 사전 계산된 테이블T에 접근하고 해당 결과값은 결과레지스터에 저장되게 된다.

〔표 6,7〕에서는 바이너리 필드상에서의 결과값에 대해 나타내고 있다. 임베디드 시스템 상에서는 프라임 필드에 비해 바이너리 필드상에서의 연산이 늦게 나타난다. 그 이유는 프라임 필드 상에 최적화된 곱셈이 하드웨어 곱셈기를 통해 소프트웨어상에서 가능하기 때문이다. 반면에 바이너리 필드는 하드웨어 상에서는 인자들간의 연산이 xor연산으로 동작하므로 보다 효율적이며 제곱연산의 경우에는 1 clock cycle만에 수행이 가능한 장점도 가진다. 최근에는 Carry-less곱셈기를 통해 보다 효율적인 연산이 퍼스널 컴퓨터에서는 가능하다. 하지만 아직 마이크로 장비에서는 성능 향상이 보다 요구된다.

〔표 6〕 ATmega128 상에서 GF(2^271) 곱셈연산

Method	Clock Cycle
Hybrid〔9〕	13,557
Comb-Window〔4〕	11,727

〔표 7〕 MSP430 상에서 GF(2^271) 곱셈연산

Method	Clock Cycle
Hybrid〔9〕	10,147
Comb-Window〔4〕	9,647

V. 결론

공개키 기반 암호화는 기존의 대칭키 기반의 암호화에 비해 키관리의 편의성과 다양한 프로토콜의 제공으로 인해 보다 각광받고 있는 암호화기술이다. 하지만 대칭키 기반 기술에 비해 요구되는 연산의 복잡도가 높아 현실적인 구현이 임베디드 장비상에서는 어려웠다. 하지만 최근들어 발표되는 연구 결과들에 의하면 임베디드 장비 상에서의 타원곡선 암호화의 실현이 보다 임박했을 뿐 아니라 임베디드 장비의 동작주파수도 30Mhz에 달할 정도로 높아지고 있다.

최근 CHES2011에서 발표된 임베디드 장비 상에서의 곱셈기의 경우 기존에 가장 효율적이라고 생각되어 왔던 Hybrid 형태의 곱셈을 대치하는 새로운 방안을 제시하여 성능을 월등히 향상 시켰다〔1〕. 임베디드 장비상에서의 곱셈기에 대한 연구는 지속적으로 개발되는 임베디드 장비의 고유한 특성과 내재된 곱셈기의 특성에 따라 그 발전 가능성이 무궁무진하다. 따라서 이에 대한 연구는 지속적으로 진행되어 가까운 미래에는 추가적인 하드웨어 구현을 대체할 수 있는 소프트웨어 구현이 가능하리라 생각된다. 본 논문에서는 지금까지 연구된 다양한 다중 곱셈기법들의 최신 연구결과를 종합 및 정리하여 나타내고 있다. 해당 Survey형식의 논문은 추후에 활발히 진행될 연구자들의 참고자료로서 활용될 수 있을 것으로 생각된다. 추후 연구는 본 논문에서 살펴본 최신 곱셈기법들의 특성을 분석하여 보다 효율적인 곱셈기법으로 발전시키는데 있다.

참고문헌

〔1〕 Michael Hutter and Erich Wenger. "Fast multi-precision multiplication for public-key cryptography on embedded microprocessors," Cryptographic Hardware and Embedded Systems - CHES 2011, vol 6917, pp. 459-474. 2011.

〔2〕 M. Shirase, Y. Miyazaki, T. Takagi, D.-G. Han, and D. Choi, "Ecient Implementation of Pairing Based Cryptography on a Sensor Node," IEICE Transactions on Information and Systems, vol 5, pp. 909-917, 2009.

〔3〕 Lopez, R, Dahab, "High-speed software

multiplication in GF(2m)," in: B.K. Roy, E. Okamoto (Eds.), First International Conference in Cryptology in India (INDOCRYPT'00), vol. 1977, pp. 203-212, 2000.

〔4〕 Leonardo B. Oliveira, Diego F. Aranha, Conrado P.L. Gouvea, Michael Scott, Danilo F. Camara, Julio Lopez, Ricardo Dahab, "TinyPBC: Pairings for authenticated idnetity-based non-interactive key distribution in sensor networks," Computer Communications, vol. 34, pp. 485-493, 2011.

〔5〕 Nils Gura, Arun Patel, Arvinderpal Wander, Hans Eberle, Sheueling Chang Shantz, "Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs," 6th International Workshop on Cryptographic Hardware and Embedded Systems, pp. 119-132, 2004.

〔6〕 A. Karatsuba and Yu. Ofman (1962). "Multiplication of Many-Digital Numbers by Automatic Computers," Proceedings of the USSR Academy of Sciences vol. 145.

〔7〕 Conrado Porto, Lopes Gouvêa, Julio López, "Software Implementation of Pairing-Based Cryptography on Sensor Networks Using the MSP430 Microcontroller," Progress in Cryptology INDOCRYPT 2009, vol. 5922, pp. 248-262, 2009.

〔8〕 S. C. Seo, D. Han, S. Hong, "TinyECCK: ecient elliptic curve cryptography implementation over GF(2m) on 8-bit MICAz mote," available at http://eprint.iacr.org/2008/122, 2008.

〔9〕 Szczechowiak, P., Kargl, A., Scott, M., Collier, M. "On the application of pairing based cryptography to wireless sensor networks," In: Proceedings of the second ACM conference on Wireless network security, pp. 1-12, 2009.

〔10〕 Atmel, "8 bit AVR Microcontroller ATmega128(L) Manual," available at http://www.atmel.com, 2004.

〔11〕 J.L. Hill, D.E. Culler, "Mica: A wireless platform for deeply embedded networks," IEEE Micro 22 vol. 6, pp. 12 - 24, 2002.

〔12〕 Texas instruments, "MSP430 Ultra-Low-Power Microcontroller," available at http://www.ti.com, 2008.

〔13〕 Texas Instruments, "The MSP430 Hardware Multiplier Function and Applications," available at http://www.ti.com, pp. 1-30, 1999.

〈著者紹介〉

서 화 정 (Hwajeong Seo) 학생회원
2010년 2월: 부산대학교 정보컴퓨터공학과 졸업
2012년 2월: 부산대학교 컴퓨터공학과 석사 졸업
2012년 3월~현재: 부산대학교 컴퓨터공학과 박사과정
〈관심분야〉 정보보안, RFID/USN, 암호 이론, VLSI 설계

김 호 원 (Howon Seo) 종신회원
1993년 2월: 경북대학교 전자공학과 졸업
1995년 2월: 포항공과대학교 전자전기공학과 석사 졸업
1999년 2월: 포항공과대학교 전자전기공학과 박사 졸업
2008년 2월: 한국전자통신연구원(ETRI) 정보보호연구단 선임연구원 / 팀장
2008년 3월~현재: 부산대학교 정보컴퓨터공학부 교수
〈관심분야〉 정보보안, 스마트그리드, RFID/USN, PKC, VLSI, Embedded System