논문 2006-43CI-2-1

다수 혹은 긴 워드 연산을 위한 레지스터 파일 확장을 통한 대칭 및 비대칭 암호화 알고리즘의 가속화

(Accelerating Symmetric and Asymmetric Cryptographic Algorithms with Register File Extension for Multi-words or Long-word Operation)

이 상 훈*, 최 린**

(Sanghoon Lee and Lynn Choi)

요 약

본 연구에서는 대칭 및 비대칭 암호화 알고리즘을 가속화하기 위해, 다수 혹은 긴 워드 연산을 위한 레지스터 파일 확장 구조 (Register File Extension for Multi-words or Long-word Operation: RFEMLO)라는 새로운 레지스터 파일 구조를 제안한다. 암호화 알고리즘은 긴 워드 피연산자에 대한 명령어를 통하여 가속화 할 수 있다는 점에 착안하여, RFEMLO는 하나의 레지스터 명을 통해 여러 개의 레지스터에 접근할 수 있도록 하여 여러 연산자에 대해 동일한 연산을 수행할 수 있도록 하거나, 여러 개의 레지스터를 하나의 데이터로 사용할 수 있게 한다. RFEMLO는 긴 워드 피연산자에 대한 명령어 집합의 추가와 이를 지원하는 기능 유닛을 추가함으로서 범용 프로세서에 적용할 수 있다. 제안된 하드웨어 구조와 명령어 집합의 효율성을 평가하기 위해 Simplescalar/ARM 3.0을 사용하여 대칭 및 비대칭의 다양한 암호화 알고리즘에 적용하였다. 실험 결과, RFEMLO을 적용한 순차적 파이프라인을 가진 프로세서에서 대칭 암호화 알고리즘의 경우 40%~160%의 성능 향상을, 비대칭 암호화 알고리즘의 경우 150% ~ 230%의 높은 성능향상을 얻을 수 있었다. RFEMLO의 적용을 통한 성능 향상은 이슈 폭의 증가를 이용한 슈퍼스칼라 구현에 따른 성능 향상과 비교할 때, 훨씬 적은 하드웨어 비용으로 효과적인 성능 향상을 얻을 수 있음을 확인하였으며 슈퍼스칼라 프로세서에 RFEMLO를 적용하는 경우에도 대칭 암호화 알고리즘에서는 최대 83.6%, 비대칭 암호화 알고리즘에서는 최대 138.6%의 추가적인 성능향상을 얻을 수 있었다.

Abstract

In this paper, we propose a new register file architecture called the Register File Extension for Multi-words or Long-word Operation (RFEMLO) to accelerate both symmetric and asymmetric cryptographic algorithms. Based on the idea that most of cryptographic algorithms heavily use multi-words or long-word operations, RFEMLO allows multiple contiguous registers to be specified as a single operand. Thus, a single instruction can specify a SIMD-style multi-word operation or a long-word operation. RFEMLO can be applied to general purpose processors by adding instruction set for multi-words or long-word operands and functional units for additional instruction set. To evaluate the performance of RFEMLO, we use Simplescalar/ARM 3.0 (with gcc 2.95.2) and run detailed simulations on various symmetric and asymmetric cryptographic algorithms. By applying RFEMLO, we could get maximum 62% and 70% reductions in the total instruction count of symmetric and asymmetric cryptographic algorithms respectively. Also, performance results show that a speedup of 1.4 to 2.6 can be obtained in symmetric cryptographic algorithms and a speedup of 2.5 to 3.3 can be obtained for asymmetric cryptographic algorithms when we apply RFEMLO to a processor with an in-order pipeline. We also found that RFEMLO can effectively improve the performance of these cryptographic algorithms with much less cost compared to issue-width increase available in Superscalar implementations. Moreover, the RFEMLO can also be applied to Superscalar processor, leading to additional 83% and 138% performance gain in symmetric and asymmetric cryptographic algorithms.

Keywords : 암호화 프로세서, 레지스터 파일 구조, 명령어 집합 구조, 암호화 알고리즘, 시뮬레이션

* 학생회원, 고려대학교 전자컴퓨터공학과
(Department of Electronics and Computer Engineering, Korea University)
** 정회원, 고려대학교 전자공학과
(Department of Electronics Engineering, Korea University)
※ 이 논문은 2002년도 한국학술진흥재단의 지원에 의하여 연구되었음 (KRF-2002-003-D00266)
접수일자: 2005년9월22일, 수정완료일: 2005년2월28일

Ⅰ. 서 론

암호화 알고리즘은 정보의 누출 및 변조등과 같은 보안 위험으로부터 시스템을 보호하는 방법으로 공유키의 사용 여부에 따라 대칭 (Symmetric) 및 비대칭 (Asymmetric) 암호화 알고리즘으로 분류된다. 대칭 암호화 알고리즘은 송/수신자가 동일한 키를 사용하여 암호화 및 복호화 과정을 진행하는 것으로, 비대칭 암호화 알고리즘에 비하여 키 길이가 짧고, 연산량이 적어 처리 속도가 비대칭 암호화 알고리즘에 비해 빠르지만, 키 공유 과정에 키의 누설 가능성이 남아있다. 복잡한 수학 문제에 기반한 비대칭 암호화 알고리즘은 암호화 및 복호화 과정에 각각 공개키(Public key)와 비밀키(Secret key)를 사용함으로써 키 공유 과정을 제거하였다. 그러나 기반이 되는 수학 문제의 복잡도 때문에 대칭 암호화 알고리즘에 비해 연산량이 많아 처리 속도가 느리다. 보안과 속도의 최적화를 위하여 대부분의 보안 프로토콜은 이 두 가지 암호화 알고리즘을 혼합하여 사용한다. 세션의 시작점에서 키 누설 가능성이 낮은 비대칭 암호화 알고리즘을 사용하여 안전하게 비밀키를 공유하고, 공유된 비밀키를 사용하는 대칭 암호화 알고리즘을 통해 전송 데이터를 빠르게 암호화 혹은 복호화 한다. 이 방식을 사용하는 대표적인 예로 IPSec[1], VPN[2], SSL[3]등이 있다.

암호화 알고리즘의 성능은 전자 결제 서버와 같은 보안 프로토콜을 사용하는 시스템의 성능에 결정적인 요소가 된다. 전자 결제 서버 시스템의 세션 길이에 대한 암호화 알고리즘 처리의 상대적 시간 비율에 대한 조사에서, 세션의 평균 크기인 21K 바이트인 경우 암호화 알고리즘의 처리 시간이 전체 처리 시간의 약 70%를 차지한다는 것을 확인하였다[8,9]. 특히 대칭 및 비대칭 암호화 알고리즘이 각각 35%의 처리시간을 차지하므로 시스템의 성능을 향상하기 위해서는 두 암호화 알고리즘 모두 가속화할 수 있는 방법이 필요하다.

이러한 암호화 알고리즘의 처리 성능을 향상시키기 위해 전용 하드웨어 혹은 명령어 추가와 같은 가속화 방안이 제안되었다. 하지만 기존의 연구는 대칭 혹은 비대칭 암호화 알고리즘 중 한 종류의 알고리즘에 대한 가속화 방안만을 제시하여 세션 처리 시간에서 대칭 암호화 알고리즘과 비대칭 암호화 알고리즘이 비슷한 비율을 차지한다는 점에서 한계를 가지고 있다.

본 연구에서는 대칭 및 비대칭 암호화 알고리즘을 가속화하기 위한 방법으로 다수 혹은 긴 워드 연산을 위한 레지스터 파일 확장 구조(Register File Extension for Multi-words or Long-word Operation: RFEMLO)라는 새로운 레지스터 구조를 제안한다. 다수 혹은 긴 워드 연산을 위한 레지스터 파일 확장 구조는 이차원 레지스터 구조에서 한 레지스터 명을 통해 정렬된 연속적인 다수의 레지스터들에 접근할 수 있도록 한다. 이러한 레지스터 사용 방식은 다수의 레지스터에 대한 SIMD 형식의 명령어를 통해 각 레지스터에 저장된 부블록(sub-block)에 대해 동일한 연산을 수행하는 대칭 암호화 알고리즘을 가속화 하고, 워드 크기보다 긴 데이터에 대한 연산을 가능하게 하여 비대칭 암호화 알고리즘을 가속화 한다. 본 연구에서는 RFEMLO를 위하여 새로운 명령어를 추가하였으며, 추가된 명령어를 실행할 수 있는 기능 유닛을 설계하였다. 이를 통해 암호화 알고리즘의 전체 실행된 명령어의 수를 대칭 암호화 알고리즘의 경우 최대 61.7%, 비대칭 암호화 알고리즘의 경우 최대 69.8% 줄일 수 있었다. 이러한 결과는 대칭 및 비대칭 암호화 알고리즘이 각각 SIMD 명령어와 긴 단어 연산 명령어를 통해 전체 명령어 수를 줄일 수 있음을 나타내며, 이러한 명령어 수의 감소는 성능향상으로 연결되어 순차적 파이프라인을 가진 프로세서에서 RFEMLO의 적용을 통해 최대 160.8%의 대칭 암호화 알고리즘 성능향상과 최대 230.5%의 비대칭 암호화 알고리즘 성능향상을 얻을 수 있었다. RFEMLO의 적용을 통한 성능 향상은 이슈 폭의 증가를 이용한 슈퍼스칼라 구현에 따른 성능 향상과 비교할 때, 훨씬 적은 하드웨어 비용으로 효과적인 성능 향상을 얻을 수 있음을 확인하였으며 슈퍼스칼라 프로세서에 RFEMLO를 적용하는 경우에도 대칭 암호화 알고리즘에서는 최대 83.6%, 비대칭 암호화 알고리즘에서는 최대 138.6%의 추가적인 성능향상을 얻을 수 있었다.

Ⅱ. 본 론

1. 관련 연구

암호화 알고리즘을 가속화하기 위한 방안으로는 전용 하드웨어를 사용하거나 명령어 추가를 통한 방법이 연구되어 왔다. 전용 하드웨어를 사용하는 경우 암호화 알고리즘의 처리속도는 빠르지만, 하드웨어를 통해 구현되지 않은 알고리즘은 처리할 수 없다는 단점이 있다. 이와 달리 명령어 추가를 통한 암호화 알고리즘의 가속화 방안은 사용자에게 새로운 알고리즘에 대한 프로그래밍의 융통성을 제공하지만, 전용 하드웨어처럼

빠른 처리를 할 수 없다는 단점이 있다. 이러한 이유로, 융통성을 유지하면서 암호화 알고리즘 처리 속도를 높이려는 연구들이 있었고, 대표적인 것으로 대칭 암호화 알고리즘을 위한 CryptoManiac[4,6]과 CRYPTONITE[5], 그리고 비대칭 암호화 알고리즘의 가속화 방안에 대한 연구로는 Intel사의 IA-64에 적용된 정수 곱셈 누적(MAC) 연산[10]과 Sun Microsystems사의 SPARC 프로세서에 적용된 이중 필드 곱셈기[11]가 있다.

CryptoManiac[4,6]에서는 8가지의 대칭 암호화 알고리즘에 사용된 명령어들 대한 세분화된 분석을 통해 대부분의 대칭 암호화 알고리즘에서 하드웨어 자원과 이슈 폭이 성능 제한의 주원인이었음을 밝혔고, 이에 대한 해결 방안으로 대표적인 대칭 암호화 알고리즘에서 많이 쓰이는 연산들에 대한 통계 조사를 바탕으로 대치(Substitution)과 치환(Permutation)을 위한 명령어를 포함한 새로운 명령어 집합을 추가하였다. 그리고 이러한 명령어들 중 간단한 명령어들은 프로세서의 클락 사이클 타임 보다 짧은 시간에 처리가 끝나며, 이로 인해 클락 사이클 타임의 비효율적인 사용을 초래한다는 점에 착안하여, 처리 시간에 따라 명령어를 'tiny', 'short', 'long' 명령어로 분류하고, 한 사이클 동안 두 개의 'tiny' 명령어 혹은 'short'과 'tiny' 명령어 조합을 처리할 수 있도록 하였다. 이와 함께, 전자 상거래 서버에서와 같이 동시에 여러 세션을 처리할 필요가 있는 시스템에서 하드웨어 자원에 의한 성능 제한을 해결할 수 있도록 멀티프로세서 구조를 제안하였다.

이와 달리 CRYPTONITE[5]에서는 자주 사용되는 명령어에 대한 통계적 조사가 아닌 알고리즘에 대한 분석을 기반으로 하여 대칭 암호화 알고리즘 소프트웨어의 근본적인 명령어 집합을 정의함으로써 CryptoManiac 보다 효율적으로 암호화 알고리즘의 특성을 프로세서 구조에 적용하고자 하였다. CRYPTONITE에서는 대부분의 대칭 암호화 알고리즘이 두 개의 경로(암호화 혹은 복호화 과정, 키 생성 과정)로 나뉘어 처리된다는 것에 착안하여 동일한 구조의 두 경로를 가진 프로세서를 설계하였다. 특히, 새로운 대칭 암호화 알고리즘 표준인 AES에 초점을 맞추어 기타 대칭 암호화 알고리즘을 가속화 할 수 있는 명령어 집합과 ALU 및 메모리 유닛을 설계하였다.

비대칭 암호화 알고리즘에 대해서는 주로 명령어 추가 방법에 대한 연구가 많이 진행되었다. 이는 비대칭 암호화 알고리즘이 복잡한 수학적 문제에 기반을 두고 있지만 실제 실행되는 명령어의 대부분이 곱셈과 덧셈이어서[7] 범용 프로세서에서 쉽게 명령어를 추가할 수 있기 때문이다. 비대칭 암호화 알고리즘은 암호화 및 복호화 과정에서의 큰 연산량이 성능저하의 원인이 되는데, 알고리즘이 기반한 수학적 모델에 따라 GF(p) 혹은 GF(2^m) 상에서의 연산으로 나뉜다[7]. 일반적으로 비대칭 암호화 알고리즘의 연산은 Montgomery 알고리즘[7]을 통해 처리되는데, Montgomery 알고리즘은 반복적인 곱셈과 덧셈을 통하여 별도의 modular 과정을 제거하여 비대칭 암호화 알고리즘을 처리한다. 두 필드 상에서의 연산과, 반복적인 곱셈과 덧셈의 특징을 가지는 비대칭 암호화 알고리즘을 가속화하기 위해 이중 필드 곱셈기(dual-field multiplier)[11,14,15]와 정수형 곱셈 누적기(integer multiply-and-adder)[10,16,17]가 제안되었다. 이중 필드 곱셈기는 약 5%의 추가적인 하드웨어를 통해 두 field의 곱셈을 모두 처리할 수 있도록 하는 것으로, GF(2^m) 상에서의 덧셈이 XOR를 통해 연산 가능하다는 점에 착안한 것이고, 정수형 곱셈 누적기는 Montgomery 알고리즘에서 항상 곱셈과 덧셈이 함께 반복적으로 처리된다는 점에서 기존의 부동 소수점 곱셈 누적기를 정수 연산에 적용한 것이다.

위와 같은 기존의 암호화 알고리즘을 가속화 하기 위한 연구들은 특정 알고리즘만을 처리하는 전용 하드웨어와 달리 다양한 알고리즘을 처리할 수 있는 융통성을 가지고 있다는 장점이 있지만, 여전히 대칭 혹은 비대칭 암호화 알고리즘 한 가지에만 초점을 맞추고 있다. 일반적인 암호화 시스템은 비대칭 암호화 알고리즘과 대칭 암호화 알고리즘이 각각의 역할을 나누어 수행하기 때문에, 시스템의 성능을 향상시키기 위해서는 대칭 및 비대칭 암호화 알고리즘 모두를 가속화 할 수 있는 방법이 필요하다. 특히 CryptoManiac과 CRYPTONITE에서 제안한 구조는 범용 프로세서의 구조와 많은 차이를 보이고 있어서 새로 설계되는 범용 프로세서에 적용하기 어렵다.

Ⅲ장에서는 이러한 문제를 해결하기 위해 제안한 다수 혹은 긴 워드 연산을 위한 레지스터 파일 확장 구조에 대해 알아보고, Ⅳ장에서는 제안된 레지스터 구조의 성능에 대한 평가를 다룬다.

2. 다수 혹은 긴 워드 연산을 위한 레지스터 파일 확장 구조 (Register File Extension for Multi-words or Long-word Operation)

이슈 폭과 하드웨어 자원 부족 문제의 해결을 위해 본 연구에서 제안한 다수 혹은 긴 워드 연산을 위한 레

지스터 파일 확장 구조(RFEMLO)는 이차원 레지스터 구조에서 정렬된 다수의 연속적인 레지스터를 하나의 레지스터 명을 통해 접근할 수 있도록 하는 레지스터 구조이다. 동일한 레지스터 명을 통해 동시에 접근되는 레지스터들을 레지스터 그룹이라 하며, 이를 위해 RFEMLO에서는 연속적인 레지스터들 중 첫 번째 레지스터 명과, 동일한 레지스터 명으로 접근하고자 하는 총 레지스터 수를 나타내는 레지스터 그룹 레벨 플래그를 통해 레지스터 파일에 접근한다. 이러한 방식은 새로운 레지스터 명명법으로 인한 명령어 형식의 변경 문제를 피할 수 있다.

다수 혹은 긴 워드 연산을 위한 레지스터 파일 확장 구조는 한 번에 접근할 수 있는 레지스터 수를 늘림으로써 이슈 폭을 증가시키지 않고도 암호화 알고리즘의 병렬성을 이용할 수 있도록 하며, 처리 가능한 데이터의 크기를 증가시켜 하드웨어 자원 부족 문제를 해결한다. 이는 대부분의 대칭 및 비대칭 암호화 알고리즘에서 문제가 되는 이슈 폭과 하드웨어 자원의 부족에 의한 성능 감소를 해결할 수 있는 방법이다. 다수 혹은 긴 워드 연산을 위한 레지스터 파일 확장 구조에는 새로운 레지스터 구조뿐만 아니라, 아래에 소개된, 레지스터 그룹을 표시하기 위한 레지스터 그룹 레벨 플래그, 제안된 레지스터 파일 확장 구조를 사용할 수 있는 새로운 명령어 집합과 이를 위한 기능 유닛을 포함한다.

- *레지스터 그룹 레벨 플래그(register grouping level flag)*: 동일한 레지스터 명을 통해 레지스터 그룹을 생성하는 총 레지스터 수를 나타낸다. 레지스터 그룹은 연속적인 $2^{(\text{레벨})}$개의 레지스터들로 구성되며, 각각의 명령어는 특정 레벨의 레지스터 그룹만 접근할 수 있도록 하였다. 레지스터 그룹 레벨 플래그는 명령어의 op-code를 분석하여 명령어 디코더에서 명령어에 따라 생성되도록 하였다. 즉 각각의 명령어가 사용하는 레지스터 레벨이 미리 정의 되어 있고, 동일한 연산을 하더라도 사용하는 레지스터 레벨에 따라 서로 다른 op-code를 할당함으로서, 명령어의 op-code의 확인만을 통해 레지스터 레벨 정보를 얻을 수 있도록 하였다. 이 방법은 명령어에 별도의 레지스터 레벨에 관한 정보를 저장할 필요가 없으므로 다수 혹은 긴 워드 연산을 위한 레지스터 파일 확장 구조가 명령어 형식에 영향을 주지 않는 장점을 가진다.

- *확장된 명령어 집합(extended instruction set)*: RFEMLO를 사용하여 워드보다 길이가 긴 데이터나 다수의 워드 데이터에 대한 SIMD 병렬연산을 하기 위한 명령어 집합이다. 다수 혹은 긴 워드 연산을 위한 레지스터 파일 확장 구조는 32bit 프로세서에서 64bit, 128bit등 워드보다 긴 데이터에 대한 연산을 가능하게 할 수 있다. 특히 멀티미디어와 같이 긴 데이터에 대한 연산이나 다수의 데이터에 대한 SIMD 연산을 주로 사용하는 응용프로그램의 가속화에 도움을 줄 수 있다. 하지만, 본 논문에서는 대칭 및 비대칭 암호화 알고리즘의 가속화만을 고려하였다.

- *확장된 기능 유닛(extended functional unit)*: RFEMLO를 사용하는 연산을 위한 추가적인 기능 유닛으로 비대칭 암호화 알고리즘을 가속하기 위한 정수형 이중 필드 곱셈 누적기 및 대칭 암호화 알고리즘을 가속하기 위한 덧셈기와 논리 연산기로 구성된다. 또한 입력 블록에 대한 암호화 연산은 확장된 명령어 집합을 통하여 처리 가능하므로 확장된 기능 유닛과 기존의 기능 유닛 간에 데이터 전송은 거의 발생하지 않는다. 그러므로 확장된 기능 유닛은 기존의 바이패스 네트워크(bypass network)와는 독립적인 바이패스 네트워크를 사용할 수 있다. 이는 레지스터 파일 확장 구조에 의한 바이패스 네트워크의 복잡도 증가를 감소시킨다.

가. 세부 구조

(1) 레지스터 파일 구조 (Register File Architecture)

RFEMLO에서 레지스터 그룹을 사용하기 위해서 레지스터 파일의 포트 수를 증가시키는 방법을 사용할 수 있으나, 이 방법은 그룹에 포함되는 레지스터의 수만큼 포트를 증가시켜야 하므로 레지스터 파일의 크기와 접근 시간을 증가시킨다[13]. 따라서, RFEMLO에서는 그림 1과 같은 이차원 구조의 레지스터 파일에서 동일 워드 선 상의 연속적인 레지스터들로 구성된 정렬된 레지스터 그룹만을 사용할 수 있도록 하여 레지스터 파일의 포트 수를 증가시키지 않고 레지스터 그룹을 사용할 수 있도록 하였다. 이 방식은 정렬되지 않은 레지스터 그룹을 사용하는 방식에 비해 서로 다른 레지스터 그룹 레벨에 속한 두 레지스터 간의 의존도 체크와 열 디코더의 구조를 간단하게 하는 장점이 있다. 표 1.은 32개의 레지스터들로 구성된 4열 레지스터 파일에서의 레지스터 그룹의 예를 보여준다.

(2) 열 디코더 및 의존도 체크 회로 (Column Decoder and Dependency Check Logic)

여러 레벨의 레지스터 그룹을 사용하는 경우 의존도 체크 회로와 레지스터 파일 내의 열 디코더는 레지스터

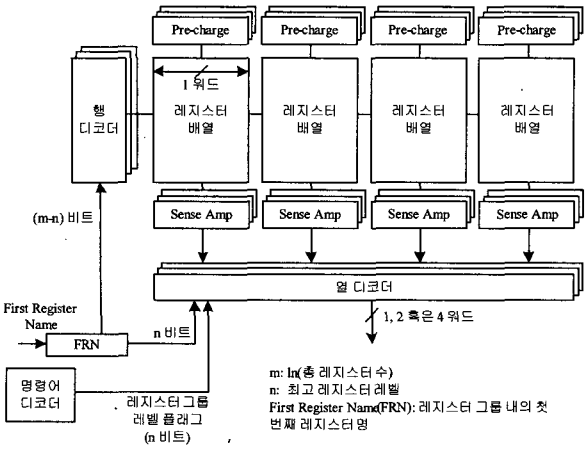

그림 1. 다수 혹은 긴 워드 연산을 위한 레지스터 파일 확장 구조를 적용한 4열 레지스터 파일 구조

Fig. 1. The 4-column register file extension for multi-words or long-word operation.

표 1. 총 32개의 레지스터에 대한 레지스터 파일 확장 구조에서의 레지스터 그룹 명명법

Table 1. Logical register names of RFEMLO architecture with 32 registers.

1 워드 (레벨 0 그룹)	2 워드 (레벨 1 그룹)	4 워드 (레벨 2 그룹)
논리 레지스터 명	논리 레지스터 그룹 명	논리 레지스터 그룹 명
R0	RT0	RTT0
R1		
R2	RT1	
R3		
R4	RT2	RTT1
R5		
R6	RT3	
R7		
...	...	...
R28	RT14	RTT7
R29		
R30	RT15	
R31		

그룹 레벨에 대한 정보를 필요로 한다. RFEMLO에서는 각 명령어가 특정 레벨의 레지스터 레벨만 사용하도록 함으로써 명령어 디코더에서 op-code를 통해 레지스터 그룹 레벨 플래그를 생성할 수 있도록 하였다. 이는 명령어에 레지스터 그룹 레벨 플래그가 포함하지 않도록 하므로 기존의 명령어 형식에 영향을 주지 않는다. 레지스터 그룹 레벨 플래그를 포함한 RFEMLO의 레지스터 파일 구조는 그림 1과 같다.

그림 1에서 알 수 있듯이 열 디코더가 레지스터 그룹 레벨 플래그를 사용하여 레지스터 그룹을 선택한다. 선택된 레지스터 그룹은 각 레벨에 따라 서로 다른 전송 경로를 가진다.

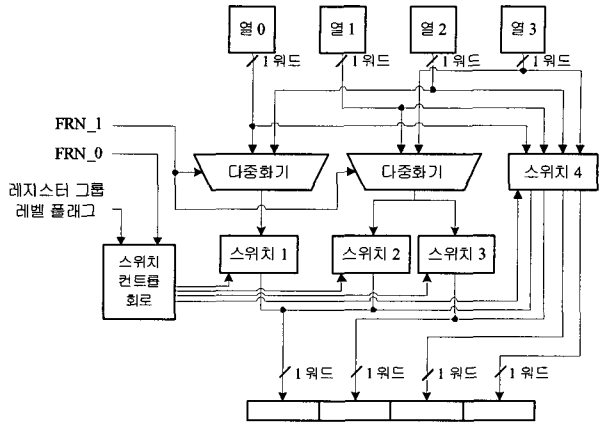

그림 2. RFEMLO가 적용된 4열 레지스터 파일에서의 열 디코더. FRN_0, FRN_1: 레지스터 그룹에 속한 첫 레지스터 명의 최하위 2 비트

Fig. 2. The column decoder of a 4 column extended register file. FRN_0, FRN_1: The LSB 2 bits of the name of first register in a register group.

그림 2는 RFEMLO가 적용된 4열 레지스터 파일을 위한 열 디코더를 보여주는 데, 레벨 0일 경우 레지스터 파일의 열에 따라 두 개의 다중화기 중 하나를 사용하고, 레벨 1일 경우 두 다중화기를 모두 사용한다. 레벨 2에서는 워드 선상의 모든 레지스터를 사용하므로 스위치를 통해 4개의 레지스터 값을 전송할 수 있도록 하였다. 스위치 1과 2는 레벨 0에서 열의 선택에 사용되며, 스위치 1과 3은 레벨 1에서 사용된다. 이 스위치들은 레지스터 그룹 레벨 플래그와 그룹에 속한 첫 레지스터 명의 최하위 bit을 이용해서 온/오프 된다.

다수 혹은 긴 워드 연산을 위한 레지스터 파일 확장 구조는 여러 레벨의 레지스터 그룹을 사용하기 때문에 서로 다른 레벨에 속한 레지스터들 간의 의존도 체크에 대한 별도의 방법이 필요하다. 기존의 의존도 체크에서는 두 레지스터 명의 동일성을 검사했지만, 레지스터 파일 확장 구조에서는 서로 다른 레벨에 속한 레지스터들 간의 포함관계에 대한 검사가 필요하다. 연속된 레지스터들로 구성된 정렬된 레지스터 그룹을 사용하는 방법은 두 레지스터 그룹의 첫 레지스터 명들의 최하위 bit들을 제외한 나머지 bit의 동일성 검사를 통해 두 레지스터 그룹의 포함관계를 간단히 검사할 수 있다. 이는 새로 추가되는 바이패스 네트워크에서 레지스터 그룹의 포함관계 검사로 인한 하드웨어 복잡도의 증가를 최소화할 수 있도록 하여 준다.

나. 확장된 명령어 집합(Extended Instruction Set)

(1) 비대칭 암호화 알고리즘 연산 분석

암호화 알고리즘은 키 운용방식에 따라 대칭 혹은 비대칭 암호화 알고리즘으로 나뉘며, 각 알고리즘의 주된 연산도 다르다. 해결하는 데 오랜 시간의 연산이 필요한 수학적 문제에 기초한 비대칭 암호화 알고리즘에서는 일반적으로 modular 연산이 가장 많이 사용된다. 비대칭 암호화 알고리즘이 기초하는 수학적 문제에 따라 Integer Factorization Problem (IFP), Discrete Logarithm Problem (DLP), Elliptic Curve Discrete Logarithm Problem (ECDLP)로 분류될 수 있으며, 대표적인 비대칭 암호화 알고리즘인 Diffie-Hellman Key Exchange, RSA, ECC는 각각 DLP, IFP, ECDLP에 기반하고 있다. Diffie-Hellman Key Exchange 와 RSA는 정수형 modular 지수 연산을 주로 사용하며, ECC는 GF(p) 혹은 GF(2^m)상에서 정의되는 Elliptic Curve 상의 점들 간의 덧셈 연산을 주로 사용한다. ECC에서 사용하는 타원 곡선상의 점들을 기하학적으로 더하는 과정은 몇 개의 modular 덧셈과 modular 곱셈 및 multiplicative inverse modulo를 필요로 한다. Montgomery method는 이러한 modular 연산을 별도의 modular 과정을 거치지 않고 곱셈과 덧셈만을 사용하여 처리할 수 있도록 하는 장점을 가지고 있어서[7] 비대칭 암호화 알고리즘의 처리에 많이 사용된다. Intel [10]과 Sun Microsystems[11]의 연구에 의하면 Montgomery method의 70% 이상이 정수형 곱셈과 덧셈이다. Montgomery method에서 이 두 연산은 쌍을 이루어 처리되며, 이를 한 명령어를 통해 처리 가능하도록 한 정수형 곱셈 누적기[10,16,17]에 대한 연구가 진행되었다. 또한 비대칭 암호화 알고리즘이 기초한 수학적 문제에 따라 정수형 또는 다항식형 두 가지 형태가 있는데, 이중 필드 곱셈기[11,14,15]는 이 두 가지 연산을 모두 가능하게 한다. 본 연구에서는 Sun Microsystems의 연구[11]에서처럼 이중 필드 곱셈 누적기를 사용하여 비대칭 암호화 알고리즘을 처리한다. 또한, 레지스터 그룹을 사용하여 이중 필드 곱셈 누적기가 워드 크기보다 큰 데이터를 처리할 수 있도록 하여 비대칭 암호화 알고리즘을 더욱 가속화 할 수 있도록 하였다.

(2) 대칭 암호화 알고리즘 연산 분석

대칭 암호화 알고리즘은 diffusion과 confusion의 개념에 기초하고 있으며, 이를 위해 표 2.에서와 같이 알고리즘 별로 다양한 연산을 사용한다.

표 2. 대칭 암호화 알고리즘의 특징 및 주 연산
Table 2. Characteristics and major operations of symmetric cryptographic algorithms.

분류		DES	RC5	IDEA	AES
블록 크기(bits)		64	2워드	64	128, variable
부 블록 크기		32	1워드	16	8
키 크기		56	최대 2048	128	128/192/256
부 키 크기		16*48	2(R+1)* 워드크기	(8*6+4) *16	R*128
라운드 수 (R)		16	0 ~ 255	9	10/12/14
주 연산	Substitution	고정값	-	-	고정값
	Permutation	고정위치	-	-	-
	Rotate	고정양	키 의존	고정양	-
	XOR	o	o	o	o
	Addition	-	mod 2^w	mod 2^{16}	-
	Multiplication	-	-	mod $2^{16}+1$	GF(2^8) 상에서의 곱셈

대칭 암호화 알고리즘에 주로 사용되는 연산은 대치(Substitution), 치환(Permutation), 논리 연산, 덧셈, 곱셈으로 분류할 수 있다. 대치 연산은 DES와 AES에서 사용되는데, 각각 6bit과 8bit의 입력, 4bit과 8bit의 출력을 가진다. 또한, 두 알고리즘 모두 미리 정의된 S-box를 사용하므로 대치 연산은 일반적인 load 연산을 통해 처리할 수 있다. Rotate와 XOR같은 논리 연산은 대부분의 알고리즘에서 사용된다. DES와 IDEA에서는 키 운용 시에 미리 정의된 양만큼의 shift를 행하며, RC5에서는 암호화 및 복호화 과정과 키 운용 시에 두 연산을 사용한다. RC5에서 사용하는 rotate는 데이터에 따라 rotate 양이 변한다는 차이점이 있다. 덧셈 연산은 RC5와 IDEA에서 사용되는데, 두 알고리즘 모두 modulo 덧셈기를 사용한다. 그런데, 이 때 사용되는 modulo값이 각각 2워드 크기와 216이므로 이는 올림수를 무시하거나, masking을 이용하여 쉽게 구현할 수 있다. 곱셈은 IDEA와 AES에서 사용되는데, IDEA에서 사용되는 곱셈은 덧셈과 같이 modulo 연산이며, base값은 (216 + 1)이다. AES에서는 GF(28)상에서 곱셈을 정의하는데, 다항식형 곱셈으로 올림수를 고려하지 않는 곱셈이다. 즉 이중 필드 곱셈기를 통해 처리 가능한 연산이다.

(3) 명령어 집합

표 3.은 대칭 및 비대칭 알고리즘의 처리를 위해

RFEMLO에 새로 추가된 명령어에 대해 정의하고 있다. 새로 추가된 모든 명령어들은 레지스터 그룹 레벨 1 혹은 2에 속하는 레지스터 그룹 중 하나를 피연산자로 사용한다. 본 연구에서 제안된 RFEMLO는 표 3.에 제안된 명령어 외에도 다양한 종류의 SIMD 형태의 명령어를 통해 멀티미디어와 같은 여러 응용프로그램을 가속화 할 수 있다. 하지만 본 연구에서는 대칭 및 비대칭 암호화 알고리즘을 가속화하기 위한 명령어 집합에 대해서만 제안하였다. 표 3.은 다수 혹은 긴 워드 연산을 위한 레지스터 파일 확장 구조에 새로 추가된 명령어 집합과 그 의미를 나타낸다. 각 명령어는 특정 레벨의 레지스터 그룹만을 사용하도록 하였다.

표 3. 추가된 명령어 집합과 그 의미
Table 3. The instruction set extension for RFEMLO.

<기호법> R - 레벨 0 레지스터, RT - 레벨 1 레지스터 그룹 (Rw와 Rx 두 개의 물리 레지스터로 구성), RTT - 레벨 2 레지스터 그룹 (Rw, Rx, Ry와 Rz 네 개의 물리 레지스터로 구성)
접미사 s - 소스 레지스터, d - 목적 레지스터
<<<, >>> 좌/우 rotation, <<, >> 좌/우 논리 shift

분류	이름	형식	기능
Load 연산	Load multiple	LW2 RTd Rs offset	연속된 2워드 데이터를 RTd에 저장
		LW4 RTTd Rs offset	연속된 4워드 데이터를 RTTd에 저장
논리 연산	And multiple with immediate	AND4 RTTd RTTs <imme> (AND2 RTd RTs <imme>)	1워드 크기의 immediate 값과 각 레지스터 값 덧셈 Rwd<-Rws&<imme> Rxd<-Rxs&<imme> Ryd<-Rys&<imme> Rzd<-Rzs&<imme>
	Shift multiple	LSL4 RTTd RTTs Rs (LSL2 RTd RTs Rs)	각 레지스터 마다 서로 다른 양의 논리 쉬프트 연산(좌) Rwd<-Rws<<(Rs) Rxd<-Rxs<<(Rs) Ryd<-Rys<<(Rs) Rzd<-Rzs<<(Rs)
		LSR4 RTTd RTTs Rs (LSR2 RTd RTs Rs)	각 레지스터 마다 서로 다른 양의 논리 쉬프트 연산(우) Rwd<-Rws>>(Rs) Rxd<-Rxs>>(Rs) Ryd<-Rys>>(Rs) Rzd<-Rzs>>(Rs)
	Rotate multiple	ROTR2 RTd RTs Rs (ROTL2 RTd RTs Rs)	오른쪽으로 회전 연산 (왼쪽으로 회전 연산) Rwd<-Rws>>>(Rs) Rxd<-Rxs>>>(Rs)
산술 연산	Addition multiple	ADD2 RTd RTs1 RTs2 (ADD4 RTTd RTTs1 RTTs2)	레지스터 그룹에 속한 각 레지스터 마다 덧셈 수행 Rwd<-Rws1+Rws2 Rxd<-Rxs1+Rxs2 Ryd<-Rys1+Rys2 Rzd<-Rzs1+Rzs2
	Multiply multiple	MULTEADD RTd RTs1 RTs2 MULTPADD RTd RTs1 RTs2	GF(2^m) 상에서의 곱셈 누적 연산 GF(p) 상에서의 곱셈 누적 연산

다. 확장된 기능 유닛(Extended Functional Unit)

이 장에서는 RFEMLO에 추가된 명령어 집합을 이용하여 암호화 알고리즘을 가속화하기 위한 기능 유닛에 대해 다룬다. 대칭 암호화 알고리즘은 일반적으로 입력 블록을 여러 개의 부 블록으로 나누어 사용하는 데, 일반적으로 2의 배수 개만큼 나누어 사용한다. 대칭 암호화 알고리즘에서 일반적으로 두 개 혹은 네 개의 부 블록에 대한 연산을 동시에 처리할 수 있기 때문에 레지스터 파일 확장 구조에서 대부분의 명령어들은 최대 레벨 2의 레지스터 그룹을 피연산자로 사용하도록 하였다. 곱셈의 경우 비대칭 암호화 알고리즘을 가속화 하고, AES의 modulo 곱셈을 처리하기 위한 것으로 레벨 1 레지스터 그룹을 소스 레지스터로, 레벨 2 레지스터 그룹을 목표 레지스터로 사용하는 이중 필드 정수형 곱셈 누적기를 사용하였다. 이중 필드 정수형 곱셈 누적기는 Sun Microsystems[11]에서 제안된 방식으로 GF(2^m)과 GF(p) 두 가지 필드에서의 연산을 처리할 수 있는 유닛이다. 그림 3a는 추가된 기능 유닛을 나타내고, 그림 3b는 기능 유닛에 포함된 이중 필드 곱셈 누적기를 간략히 보여주고 있다. 이중 필드 곱셈 누적기를 구성하는 곱셈기와 덧셈기는 정수형 곱셈기 및 덧셈기에 XOR를 추가하여 구현할 수 있다[11]. 논리 연산기와 덧셈기는 대칭 암호화 알고리즘을 위한 명령어가 최대 레벨 2의 레지스터 그룹을 사용하기 때문에 각각 4

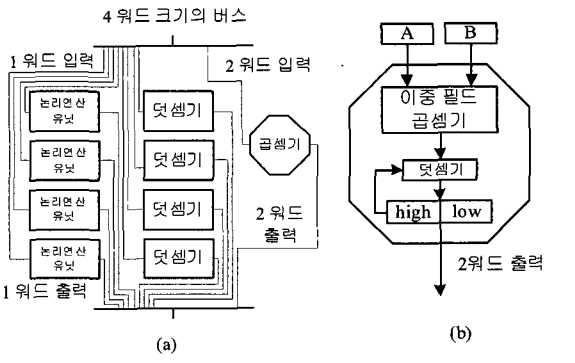

그림 3. 확장된 기능 유닛과 그에 포함된 이중 필드 곱셈 누적기
Fig. 3. Extended functional unit and a dual field multiply-and-adder.

개씩 포함하였다.

라. 하드웨어 비용(Hardware Cost)

레지스터 파일 확장 구조에서는 레지스터 파일, 의존도 체크 회로, 기능 유닛 그리고 바이패스 네트워크 회로에서 추가적인 하드웨어를 필요로 한다. 우선 이차원 구조의 레지스터 파일에서 레지스터 그룹을 지원하기 위한 열 디코더가 필요하다. 레지스터 파일 확장 구조에서 사용된 열 디코더에서 각각의 데이터 경로에 포함되는 다중화기와 스위치는 ln(최대 레벨 값)에 비례하여 증가한다. 이는 데이터 경로가 다중화기 하나로 이루어져있던 기존의 열 디코더에 비해 레지스터 파일의 접근 시간을 증가시키는 요인이 된다[13].

바이패스 네트워크 회로에서는 레지스터 파일 확장 구조에서 레지스터 그룹을 사용하므로 바이패스 경로의 폭을 넓혀야 한다. 그런데, 암호화 알고리즘의 입력 블록에 대한 직접적인 연산은 추가된 기능 유닛만으로 충분하기 때문에 기존의 바이패스 네트워크와 별개의 바이패스 네트워크를 사용하여 추가된 기능 유닛과 연결할 수 있다. 이는 기존의 바이패스 네트워크의 폭을 넓히지 않아도 되므로 바이패스 네트워크의 복잡도 증가를 줄일 수 있다. 다만 레지스터 그룹의 사용으로 인해 바이패스 제어 회로에서 레지스터 그룹 레벨에 따른 포함관계에 대해 검사해야 하므로 바이패스 제어 회로가 복잡해지지만, [18]에 따르면 바이패스 제어 회로는 전체 바이패스 네트워크의 성능에 큰 영향을 주지 않으므로 무시할 수 있다.

III. 실 험

제안된 하드웨어 구조와 명령어 집합의 효율성을 증명하기 위해 Simplescalar/ARM 3.0(with gcc 2.95.2)[19]과 최적화된 코드를 사용하였다. Simplescalar/ARM 3.0은 ARM 프로세서의 명령어 집합에 기반을 두고 있다. 본 논문에서는 제안된 새로운 명령어를 Simplescalar/ARM과 크로스 컴파일러에 추가하였다. Simplescalar/ARM의 기본 프로세서 모델은 in-order pipeline으로 설정하였고, 분기 예측은 암호화 알고리즘에 거의 영향을 주지 않으므로[4,6] 완전 분기 예측(perfect branch prediction)으로 설정하였다. 본 논문에서는 대표적인 대칭 암호화 알고리즘인 DES, AES와 IDEA, 비대칭 암호화 알고리즘인 RSA와 GF(2^m) 상에서의 ECC에 대하여 실험하였으며, 대칭 암호화 알고리즘은 가장 보편적으로 사용되는 Cipher-block Chaining(CBC) 모드를 사용하였다.

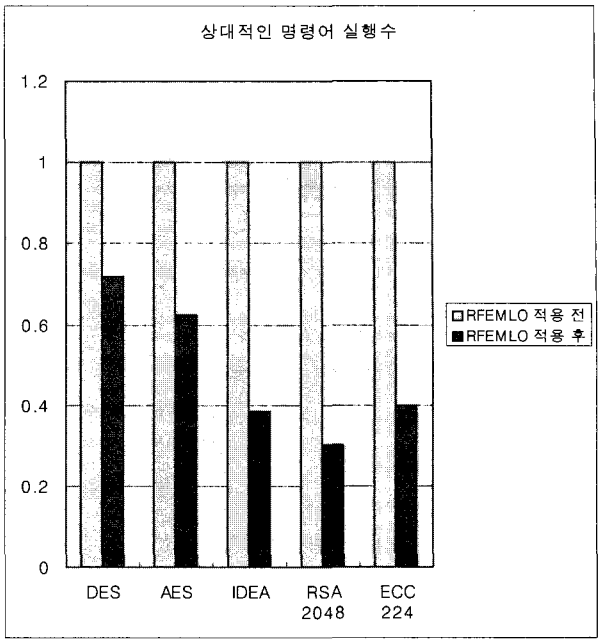

그림 4. 암호화 알고리즘 별 실행 명령어 수 감소 비율
Fig. 4. Relative instruction count of cryptographic algorithms.

그림 4는 RFEMLO 구조에서 각 알고리즘의 총 실행 명령어 감소 비율을 나타낸다. 실험에 따르면 대칭 암호화 알고리즘의 경우 최대 61.7%, 비대칭 암호화 알고리즘의 경우 최대 69.8%로 명령어 수를 감소시킬 수 있었다. DES의 명령어 감소 비율이 가장 낮은 이유는 DES 알고리즘이 하드웨어로 구현하기 쉽도록 설계되었기 때문에 프로세서 상에서 소프트웨어 루틴으로 처리하는 경우 성능향상이 가능한 부분이 많지 않기 때문이다. 특히 DES 알고리즘의 대부분을 차지하는 대치(Substitution)와 치환(Permutation)은 테이블을 이용하여 구현되고, 이러한 데이터 전송관련 명령어는 다수 혹은 긴 워드 연산을 위한 레지스터 파일 확장 구조를 통해 SIMD형태의 명령어로 치환하기 힘들다. AES는 DES와 달리 소프트웨어로 쉽게 구현하고 처리할 수 있도록 설계된 알고리즘이지만, AES 역시 데이터 전송과 관련한 명령어들이 큰 비율을 차지하고 있어서 레지스터 파일 확장 구조에 의한 실행 명령어 수 감소비율이 높지 않다. 이에 비해 IDEA는 주로 산술연산과 논리연산이 많아서 다수 혹은 긴 워드 연산을 위한 레지스터 파일 확장 구조를 통한 실행 명령어 수 감소비율과 성능개선 비율이 높게 나타난다. 특히 곱셈과 덧셈이 대부분을 차지하는 RSA와 ECC는 제안된 구조를 통해

많은 실행 명령어 수 감소와 높은 성능 향상을 이룰 수 있었다. 이는 하드웨어 자원만 충분하다면 RSA와 ECC에서 반복적으로 처리되는 정수형 곱셈 누적 연산을 충분히 가속화할 수 있기 때문이다.

그림 5는 수정 전의 코드와 다수 혹은 긴 워드 연산을 위한 레지스터 파일 확장 구조를 적용 후 수정된 코드에서 각 대칭 암호화 알고리즘의 주요 연산들 - 메모리 관련 연산, 산술 연산, 논리 연산 - 의 비율을 보여준다. 세 알고리즘 모두 산술 연산과 논리 연산에서 큰 감소를 나타냈는데, 이는 레지스터 파일 확장 구조에서 추가된 명령어들이 대부분 산술 연산과 논리 연산이기 때문이다. 그림 5의 실험 결과에 따르면 대칭 암호화 알고리즘의 경우 추가된 명령어들로 인한 연산 종류별 명령어 수 감소율이 산술 연산은 평균 58.3%, 논리 연산은 평균 70.2%로 나타났다. 각 알고리즘 중에서는 IDEA의 감소율이 가장 높게 나타났는데, 이는 IDEA가 4개의 부 블록에 대해 산술 연산과 논리 연산을 반복적으로 실행하는데, 이 중 대부분이 SIMD 형태의 명령어들로 처리 가능하며, 산술 연산과 논리 연산이 병렬적으로 처리 가능한 경우가 많기 때문이다.

순차 프로세서에 다수 혹은 긴 워드 연산을 위한 레지스터 파일 확장 구조의 적용 전후의 처리 속도에 대한 상대적인 비교 결과가 그림 6에 나타나 있다. 결과에 따르면 최대 160.8%의 대칭 암호화 알고리즘 성능향상과 최대 230.5%의 비대칭 암호화 알고리즘 성능향상을 얻을 수 있었다. 비대칭 암호화 알고리즘의 성능 증가 비율이 더욱 높게 나타나는 이유는 RFEMLO를 통해 워드 최대 69.8%로 감소시킬 수 있었기 때문이다. 이러한 장점은 단일 순차 프로세서의 이슈 폭을 증가시킨 경우보다 다수 혹은 긴 워드 연산을 위한 레지스터 파일 확장 구조만 적용 시킨 경우에도 잘 드러나는데, 다수 혹은 긴 워드 연산을 위한 레지스터 파일 확장 구조만 적용 시킨 경우 한 번에 처리할 수 있는 데이터의 양이 두 배로 늘어나며, 곱셈 누적기의 사용을 통해 한 명령어로 곱셈 연산과 덧셈 연산을 처리하였기 때문이다. 반면, 대칭 암호화 알고리즘에서는 성능 증가가 알고리즘에 따라 많은 차이를 보였다. DES의 경우 4-way 슈퍼스칼라 순차 프로세서에 RFEMLO를 적용한 경우 약 5%의 성능 증가를 보여 가장 낮은 비율을 보였다. 이는 DES 알고리즘이 4-way 슈퍼스칼라 순차 프로세서에서 성능 감소가 거의 발생하지 않는 다는 것을 뜻한다. 반면 AES와 IDEA에서는 4-way 슈퍼스칼라 순차 프로세서에 RFEMLO를 적용한 경우 각각 31.5%, 83.6%의 추가적인 성능향상을 얻을 수 있었다. 특히 IDEA에서 성능향상 비율이 높은 것은 IDEA 알고리즘이 4개의 부 블록에 대한 반복적인 논리 연산으로

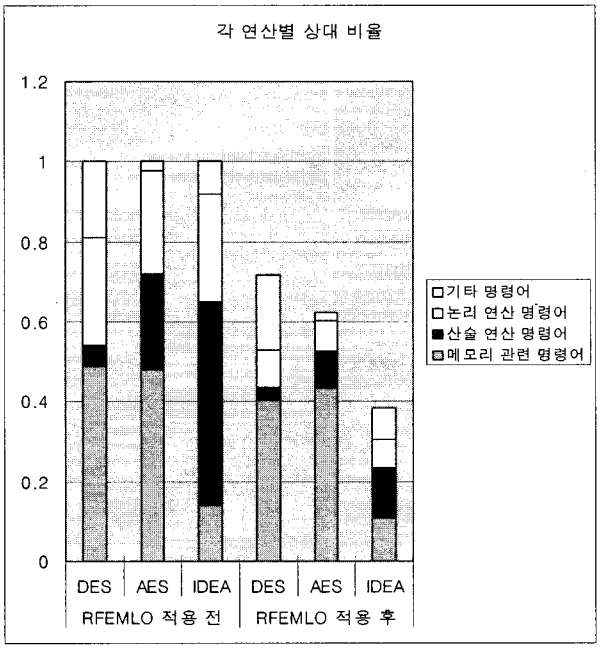

그림 5. 각 대칭 암호화 알고리즘의 연산 종류별 명령어 분포

Fig. 5. Distribution of instruction types in cymmetric cryptographic algorithms.

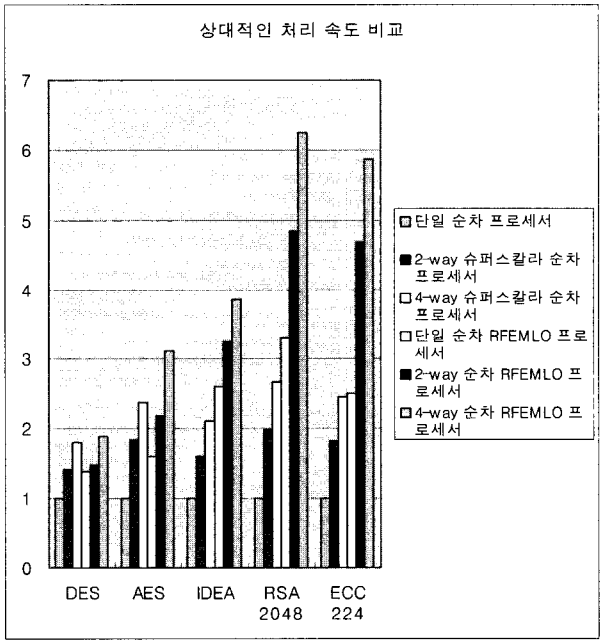

그림 6. RFEMLO와 슈퍼스칼라 구현에 따른 각 알고리즘 별 성능 비교

Fig. 6. Comparision of relative speedup by applying RFEMLO and superscalar implementation to various cryptographic algorithms.

이루어져 있으며, 이 연산 대부분이 SIMD 형태의 연산을 통해 가속화되었기 때문이다. 또한 비대칭 암호화 알고리즘인 RSA와 ECC에서는 각각 132.8%, 138.6%의 추가적인 성능향상을 얻을 수 있었다. 이는 비대칭 암호화 알고리즘의 경우 워드보다 긴 단어에 대한 연산을 수행하기 때문에 단일 연산을 통해 처리되는 데이터가 커질 수 록 처리 속도는 증가하기 때문이다.

그림 7은 기존 연구와 RFEMLO의 정성적 성능 비교 결과이다. 3DES 알고리즘의 경우 CRYPTONITE가 약 5.9배의 성능 향상을 보여 CryptoManiac과 RFEMLO에 비해 높은 성능 향상을 보였는데, 이는 CRYPTONITE에서 DES 모듈을 별도로 추가하였기 때문이다. 그러나 AES와 IDEA의 경우, RFEMLO는 각각 3.13, 3.857 배의 성능 향상을 보여 각각 2.25, 2.13 배의 성능 향상을 보인 CryptoManiac과 각각 2.8, 3 배의 성능 향상을 보인 CRYPTONITE에 비해 높은 성능을 보임을 확인할 수 있다. 또한, 기존 연구는 부 프로세서 형태의 구조를 제안한 반면 RFEMLO는 범용 프로세서에 적용 가능한 장점이 있다.

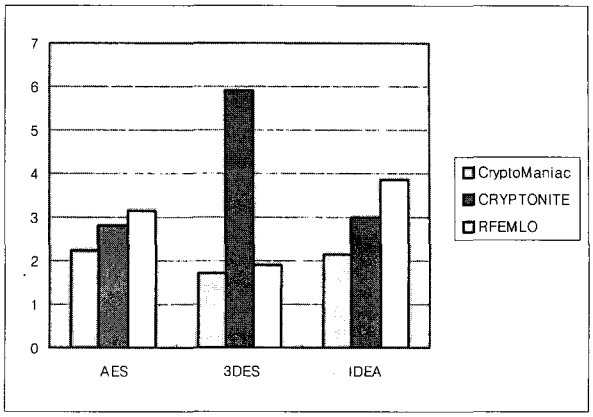

그림 7. 대칭 암호화 알고리즘에 대한 기존 연구와 RFEMLO의 성능 향상에 대한 정성적 비교

Fig. 7. Qualitative comparison of relative speedup between RFEMLO and conventional works.

IV. 결 론

기존의 암호화 알고리즘 가속화 방안에 대한 연구는 대칭 혹은 비대칭 암호화 알고리즘에 국한되어 연구되었다. 또한, 대칭 암호화 알고리즘의 경우 범용 프로세서가 아닌 보조 프로세서 형태에 대한 연구가 대부분이었으며, 이러한 연구에서 제안된 방법들은 범용 프로세서와 상이한 부분이 많아 새로 설계되는 프로세서에 적용하기 힘들다는 단점이 있었다. 암호화 세션길이의 증가로 인한 대칭 및 비대칭 암호화 알고리즘의 처리 능력이 전체 시스템의 성능에 큰 영향을 끼친다는 점을 고려할 때, 대칭 및 비대칭 암호화 알고리즘 모두를 가속화 할 수 있는 새로운 구조가 필요하다. 이를 위해 RFEMLO는 워드 크기보다 큰 데이터와 SIMD 형태의 연산을 지원하는 명령어 집합과 기능 유닛의 추가를 통해 범용 프로세서에서의 대칭 및 비대칭 암호화 알고리즘의 처리 성능을 가속화 한다. 또, RFEMLO를 위하여 추가된 기능 유닛에서 바이패스 네트워크를 별도로 구현함으로써 기존 하드웨어의 복잡성을 증가시키지 않고 기능 유닛을 설계하였으며 명령어 집합 역시 기존 명령어 형식의 변화없이 적용할 수 있도록 설계하였다.

제안된 RFEMLO의 성능 평가를 위하여 Simplescalar3.0/ARM과 다수의 대칭 및 비대칭 암호화 알고리즘을 사용하여 자세한 시뮬레이션 분석을 수행하였으며 실험 결과, RFEMLO는 대칭 및 비대칭 암호화 알고리즘 모두에 상당한 성능향상을 가져오는 사실을 확인하였다. 특히 SIMD 형태의 명령어에 적합한 IDEA와 워드 크기보다 큰 데이터에 대한 연산을 필요로 하는 비대칭 암호화 알고리즘들에서 상대적으로 높은 성능 향상을 나타냈다.

RFEMLO,는 암호화 알고리즘은 물론 다수의 워드 연산 또는 긴 데이터 연산을 사용하는 멀티미디어, 신호 처리, 그래픽 등 다양한 응용 프로그램에서 상당한 성능향상을 얻을 수 있을 것으로 기대하며 앞으로 이에 대한 보다 심도 깊은 연구가 필요할 것으로 사료된다.

참 고 문 헌

[1] R. Atkinson. "Security architecture for the internet protocol." IETF Draft Architecture ipsec-arch-sec00, 1996.

[2] Robert Moskowitz, "What is a Virtual Private Network?" http://www.networkcomputing.com/905/905colmoskowitz.html

[3] The SSL Protocol, version 3.0, Netscape, Inc, http://home.netscape.com/eng/ssl3/draft302.txt, 1999.

[4] Lisa Wu, Chris Weaver, Todd Austin, "CryptoManiac: a fast flexible architecture for secure communication", International Conference on Computer Architecture, Proceedings of the 28th annual International Symposium on Computer Architecture, pp. 110-119, 2001.

[5] Rainer Buchty, Nevin Heintze, Dino Oliva,

"Cryptonite - A Programmable Crypto Processor Architecture for High-Bandwidth Applications", International Conference on Architecture of Computing Systems, ARCS 2004, LNCS 2981, pp. 184-198, 2004.
[6] J. Burke, J. McDonald, and T Austin. "Architectural Support for Fast Symmetric-Key Cryptography". Proceedings of ASPLOS, 2000.
[7] Alfred J. Menezes, Paul C. van Oorschotand, Scott Al Vanstone, Handbook of Applied Cryptography, CRC press
[8] M. Arlitt, C. Williamson, "Web server workload characterization: The search for invariants", Proceedings of the ACM SIGMETRICS '96 Conference, April, 1996.
[9] Cristian Coarfa, Peter Druschel and Dan S. Wallach, "Performance Analysis of TLS Web Servers", In Proceedings of The Ninth Network and Distributed System Security Symposium (NDSS 02), February, 2002.
[10] Stephen Moore, "Enhancing Security Performance Through IA-64 Architecture", Intel Corporation
[11] Hans Eberle, Sheueling Chang Shantz, Vipul Gupta, Nils Gura, "Accelerating Next-generation Public-key Cryptography on General-purpose CPUs", Hot Chips 16, Stanford, Aug, 2004.
[12] J. H. Hong, C. W. Wu, "Radix-4 modular multiplication and exponentiation algorithms for the RSA public-key cryptosystem", Design Automation Conference (ASP-DAC 2000), pages 565-570, 2000.
[13] Subbarao Palacharla, Norman P. Jouppi, J. E. Smith, "Complexity-Effective Superscalar Processors", In 24th International Symposium on Computer Architecture, pages 206-218, June 1997.
[14] A. Satoh, K. Takano, "A Scalable Dual-Field Elliptic Curve Cryptographic Processor", IEEE Transactions on Computers, vol. 52, no.4, April 2003, pp 449-460
[15] E. Savas, A. F. Tenca, C. K. Koc, "Dual-field multiplier architecture for cryptographic applications", Thirty-Seventh Asilomar Conference on Signals, Systems, and Computers, pp 374-378, IEEE Press, Pacific Grove, California, November 9-12, 2003.
[16] Johann Grobschadl, Guy-Armand Kamendje, "Instruction set extension for fast elliptic curve cryptography over binary finite fields GF(2m)", IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP'03), June 2003.
[17] J. Grobschadl, "Instruction set extension for long integer modulo arithmetic on RISC-based smart cards", 14th Symposium on Computer Architecture and High Performance Computing (SCAB-PAD'02), October 2002 pp 13-19
[18] P. S. Ahuja, D. W. Clark, and A. Rogers. "The performance impact of incomplete bypassing in processor pipelines", In Proceedings of the 28th Annual International Symposium on Microarchitecture, 1995.
[19] SimpleScalar Toolset ver. 3.0 http://www.simplescalar.com

저 자 소 개

이 상 훈(학생회원)
2004년 고려대학교 전기전자전파 공학과 학사 졸업.
2004년~현재 고려대학교 전자 컴퓨터공학과 석박사 통합 과정 재학.

<주관심분야 : Application Specific Instruction Processors, Secure Processor Architectures, Sensor Networks and Ubiquitous Computing, Network Protocols and Algorithms>

최 린(정회원)
1986년 서울대학교 컴퓨터공학과 학사 졸업.
1988년 서울대학교 컴퓨터공학과 석사 졸업
1996년 University of Illinois at Urbana-Champaign 컴퓨터 공학과 박사 졸업.
1988년~1990년 한국통신 연구개발단 전임연구원
1996년~1998년 Intel Corporation, Itanium CPU Design, Sr. Design Engr.
1998년~2000년 University of California, Irvine 조교수
2000년~2002년 고려대학교 전자공학과 조교수
2002년~현재 고려대학교 전자공학과 부교수
<주관심분야 : Application Specific Instruction Processors, Secure Processor Architectures, Sensor Networks and Ubiquitous Computing, Network Protocols and Algorithms>