스마트 카드용 Rijndael 암호 프로세서의 ASIC 설계

이윤경, 이상우, 김영세

한국전자통신연구원 IC카드 연구팀

ASIC Design of Rijndael Processor for Smart Card

Yun-Kyung Lee, Sang-woo Lee, Young-Se Kim

IC Card Research Team, ETRI

요 약

정보화의 부작용이라 할 수 있는 보호되어야 할 자료의 유출이 심각해지면서 특정 사용자간에 데이터를 암호화해서 전송하고 다시 복호화 해서 데이터를 얻어야 할 필요성이 커지고 있다. 따라서 데이터의 고속 암호화 및 복호화 기술의 개발이 시급하다.

데이터의 암호화 및 복호화를 위해서는 비밀키 암호를 사용하는데 대표적인 비밀키 암호로 Rijndael(AES) 암호가 있다. Rijndael 암호는 기존의 블록암호와는 달리 비교적 적은 게이트 수를 사용하여 하드웨어로 구현할 수 있고, 키 관리와 암/복호 속도 측면에서 하드웨어 구현이 소프트웨어 구현보다 우수하기 때문에 코프로세서 형태로 구현하여 스마트카드, SIM카드, 모바일 시스템 등에 적용할 수 있다.

본 논문에서는 스마트 카드에 적합한 Rijdnael 암호 프로세서의 구현 방법에 관하여 기술하였다. 본 논문에서 제시한 방법으로 구현하여 Synopsys로 합성하여 18000 게이트 정도의 적은 게이트 수로 3GHz를 넘어서는 동작 주파수, 2.56 Gbps의 높은 암/복호율을 갖는 프로세서의 구현이 가능함을 확인할 수 있었다.

Ⅰ.서론

정보화 시대를 살아가면서 많은 데이터를 접하게 되고 또한 자신의 정보도 더욱 많은 사람과 공유하게 됨에 따라서 특정 사용자간에 데이터를 암호화해서 전송하고, 암호화된 데이터를 받아서 복호화 하여 정보를 사용해야할 필요성이 점점 커지고 있다. 따라서 많은 양의 데이터를 빠른 시간내에 암호화 및 복호화 하기 위하여 비밀키 암호의 하드웨어 구현이 증가하고 있다.

비밀키 암호 알고리듬의 대명사였던 DES의 안전성에대한 의심에서 미국 상무부 기술 표준국 NIST(National Institute of Standard and Technology)에서 AES (Advanced Encryption Standard) 알고리듬을 공모하였고, 그 결과 2001년 Rijndael 알고리듬이 AES 알고리듬으로 최종 선택되어 미국의 표준 암호로 자리잡게 되었다 [1]. Rijndael 알고리듬은 128 bits, 192 bits, 256 bits의 블록 데이터에 대해서 128 bits, 192 bits, 256 bits 키를 사용하여 암호화 및 복호화를 하며, 현재 128 bits 블록 데이터에 대한 암호 및 복호 알고리듬만이 표준으로 정해져 있다. 그리고 알려진 모든 공격에 강하고 취약키가 존재하지 않는 것으로 알려져 있다. 또한 이전의 블록 암호들이 소프트웨어 구현성만을 고려하여 고안된 것과는 달리, AES는 소프트웨어 구현 뿐만아니라 하드웨어 구현까지 염두에 둔 구조로 되어있다 [2].

AES는 하드웨어 및 소프트웨어로 구현 가능하지만, 소프트웨어로 구현할 경우 데이터의 암호화 및 복호화에 시간이 많이 걸리고 해킹에 의해 키가 쉽게 노출될 수 있기 때문에 주로 하드웨어로 구현한다. AES 알고리듬을 하드웨어로 구현하여 코프로세서 형태로 스마트카드, SIM(Subscriber Identity Module) 카드, 모바일 시스템들에 적용함으로써 보호되어야 할 데이터의 고속 암호화 및 복호화에 적용할 수 있다.

본 논문에서는 Rijndael 암호 알고리듬의 구조와 알고리듬의 하드웨어 구현에 관하여 논의할 것이다.

Ⅱ. Rijndael 알고리듬

Rijndael 알고리듬은 입력 데이터의 크기와 사용하는 키 길이에 따라 각각 다른 수의 라운드 연산을 통해 데이터를 암/복호화 한다. 입력 데이터 128 bits에 대해서 128bits, 192 bits, 256 bits의 세 가지 키 길이에 따라서 10/12/14 회의 라운

드 연산을 수행한 후 암/복호화된 데이터를 얻는다.

암호화용 라운드 함수는 Substitution, Shift_Row, MixColumn, Add_Round_Key의 네 가지 변환을 차례로 수행하고, 복호화용 라운드 함수는 암호화용 라운드 함수의 역변환으로 Inverse Shift_Row, Inverse Substitution, Add_Round _Key, Inverse MixColumn의 네 가지 변환을 차례로 수행한다. 암호화와 복호화 모두 라운드 함수를 수행하기 전에 입력 데이터와 입력 키와의 XOR 연산을 수행한다. 그 결과 데이터는 첫 번째 라운드 함수의 입력 데이터가 되며, 라운드수 -1번의 라운드 함수 연산을 수행한 후 마지막 라운드는 각 라운드 함수 연산에서 MixColumn(복호화 연산의 경우 Inverse MixColumn) 변환을 제외한 나머지 세 가지 변환을 차례로 수행한다.

라운드 함수를 수행할 때 각 라운드의 결과, 즉 암/복호화의 중간 결과를 "state"라 하고 이는 8 비트를 하나의 원소로 하여 4개의 열을 갖는 행렬 형태이다. 128 비트 데이터의 경우 4열 4행의 행렬 형태의 "state"가 된다. 외부 입력이 최상위 비트(MSB)부터 들어온다고 가정할 때 입력 데이터의 "state"로의 매핑은 그림 1과 같다.

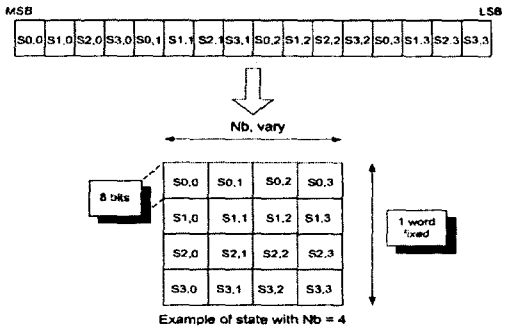

그림 1: 입력 데이터의 "state" 매핑

라운드 연산을 수행할 때 매 라운드마다 라운드 키가 필요하다. 따라서 128 bits 크기의 데이터와 키를 고려할 때 10회의 라운드 연산이 필요하고 라운드 연산을 시작하기 전에 입력키와 데이터의 XOR연산 과정이 있으므로 총 11개의 키가 데이터 암/복호화에 사용된다. 따라서 16 Bytes * 11 = 176 Bytes의 키가 필요하다. 입력키가 들어왔을 때 라운드 연산에 필요한 라운드 키를 미리 생성하여 저장 공간에 저장해 두는 방법을 사용할 수도 있고, 라운드 연산을 수행하면서 동시에 각 라운드에서 필요한 라운드키를 생성하는 on-the -fly 방식으로 키를 생성할 수도 있다. 전자의 경우 저장 공간이 넉넉한 시스템에서 아주 많은 데이터를 동일한 키를 사용하여 암호화 또는 복호화 할 때 적용할 수 있고, 후자는 제한된 저장 공간과 데이터의 빠른 암호화 및 복호화를 요구하는 시스템에 적용할 수 있다. 또한 전자에 비해 후자가 훨씬 적은 크기의 하드웨어 구현이 가능하고 키가 외부에 노출될 가능성이 적어지기 때문에 본 논문에서 제시한 Rijndael 암호 프로세서의 구현에는 후자의 방법을 적용하였다.

그림 2는 on-the-fly 방식의 키 생성 모듈을 포함한 라운드 연산의 흐름도를 보여준다. 그림 2에서 'In_data'는 라운드 입력 데이터를, 'mode'는 암호화(0) 또는 복호화(1)의 선택을, 'preKey'는 라운드의 입력키, 'RK'는 라운드 연산에서 사용될 라운드 키를 나타낸다.

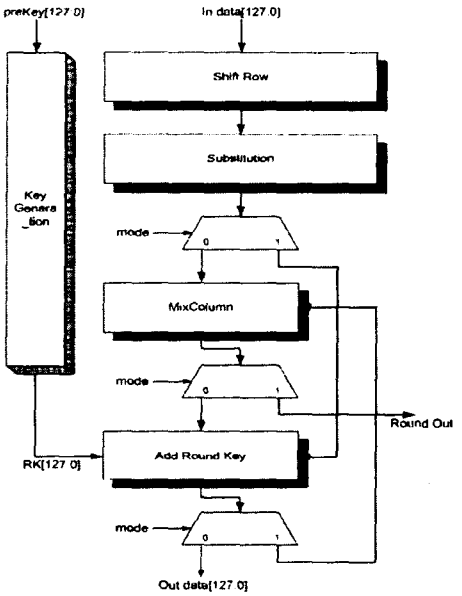

그림 2: 키 생성 모듈을 포함한 라운드 연산

Ⅲ. 라운드 연산의 하드웨어 구현

1. Substitution/Inverse Substitution 변환

Substitution(Inverse Substitution) 변환은 S-box(SI-box)를 이용하여 1 바이트의 입력 데이터를 1 바이트의 출력 데이터로 변환하는 비선형 바이트 매핑 연산이다. S-box는 $GF(2^8)$에서의 곱셈에 대한 역원을 구한 후(0은 0으로 매핑된다고 가정한다.), 그 결과값에 Affine Transformation을 행한 결과값들로 이루어진 역변환 가능한 256 바이트의 substitution table 이다.

$GF(2^8)$에서의 곱셈에 대한 역원은 table 형태로 구현할 수 있고[3] (이 table은 [4]를 참고하자.), 또한 직접 계산할 수도 있는데 Extended Euclidean algorithm[5]은 하드웨어 구현에 부적절하고, 수행 시간도 많이 걸리므로 $GF(2^8)$의 값

을 GF(2^4)의 값으로 매핑하여 GF(2^4)에서의 곱셈에 대한 역원을 구한 후, 다시 GF(2^8)의 값으로 매핑하는 방법을 사용하여 구현할 수 있다[4]. 1

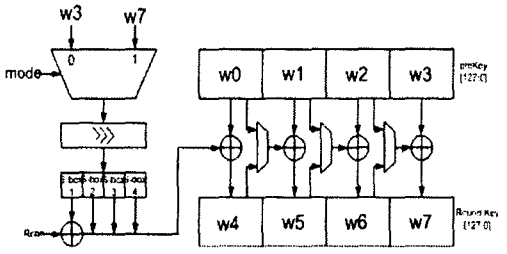

그림 3: 128 비트 키 생성 모듈

바이트의 입력 데이터에 대한 Affine transformation은 해당 입력 비트의 XOR연산과 inverse 연산을 사용하여 각 출력 비트의 값을 구하는 방법으로 구현하였다.

SI-box는 앞서 설명했듯이 S-box의 역변환으로, SI-box는 바이트 입력에 대해서 Affine transformation의 역변환을 수행한 후, 그 결과값에 GF(2^8)에서의 곱셈에 대한 역원을 취함으로써 얻을 수 있다. Affine transformation의 역변환은 Affine transformation의 하드웨어 구현과 마찬가지로 해당 입력 비트의 XOR 연산과 inversion을 통해서 각 출력 비트의 값을 구하는 방법으로 구현하였다.

S-box(SI-box)를 lookup table 형태로 구현한 결과와 위에서 설명한 방법으로 구현한 결과를 synopsys에서의 합성을 통해 검증한 결과 S-box를 lookup table이 아닌 직접 계산하는 방법으로 구현하는 것이 하드웨어 리소스(게이트 수)를 약 33% 적게 차지하는 것을 알 수 있었다.

2. Shift_Row / Inverse Shift_Row 변환

Shift_Row 연산은 라운드 연산의 다른 세 연산과 달리 "state" 의 행을 기본으로 이루어지는 변환이다. 각 행을 바이트 단위로 일정 수만큼 쉬프트 하는 연산으로 Shift_Row 변환은 오른쪽으로 순환 쉬프트 연산을 수행하고, Inverse Shift_Row 변환은 왼쪽으로 순환 쉬프트 연산을 수행한다. 첫번째 행은 쉬프트가 없고, 두번째 행은 1 바이트, 세번째 행은 2 바이트, 네번째 행은 3 바이트만큼의 순환 쉬프트를 한다.

Shift_Row와 Inverse Shift_Row 변환에서 첫번째 행과 세번째 행은 하드웨어 공유가 가능하고 또한 와이어 연결 만으로 구현할 수 있다. 두번째 행과 네번째 행은 암호화 과정과 복호화 과정을 구분하는 2 입력 멀티플렉서(MUX)를 사용하여 와이어 연결을 통해 구현한다. 따라서 암호화와 복호화 모두 가능한 128 비트 Shift_Row와 Inverse Shift_Row 변환기를 모두 구현하는데 8 개의 멀티플렉서(MUX) 만을 이용한다.

3. MixColumn / Inverse MixColumn 변환

MixColumn / Inverse MixColumn 변환은 " state" 의 각 열(column)을 4항 다항식으로 고려할 때 이 식과 고정된 다항식a(x)와의 x^4+1 모듈리 곱셈 결과를 취하는 변환이다. MixColumn 변환의 고정된 다항식 a(x)는 식(1)과 같고, Inverse MixColumn 변환의 고정된 다항식 $a^{-1}(x)$는 식(2)와 같다.

$$a(x)=\{03\}x^3+\{01\}x^2+\{01\}x+\{02\} \quad -- (1)$$

$$a^{-1}(x)=\{0b\}x^3+\{0d\}x^2+\{09\}x+\{0e\} \quad -- (2)$$

MixColumn / Inverse MixColumn 모듈의 하드웨어 구현은 입력 데이터에 {03}, {02}, {0b}, {0d}, {09}, {0e}를 곱하는 다항식 곱셈기를 각각 구현하였는데, 암/복호 모드에 따라 각 곱셈기의 입력 데이터를 제어함으로써 암/복호화용 MixColumn 연산 모듈을 구현할 수 있다. 곱셈기의 구현은 Xtime 함수[1]를 응용한 것으로 수십개의 XOR 게이트와 Inverter만을 사용하여 구현가능하다.

Ⅳ. 키 생성 모듈의 하드웨어 구현

Rijndael 암호는 암호화와 복호화에 사용되는 키의 생성 방법이 다르므로 1 바이트 쉬프트 연산을 수행하는 Rotate 변환 모듈과 S-box , 현재의 라운드 수에 따라서 다른 8 비트의 값을 내보내는 RconReg 모듈은 암호화와 복호화용 키 생성시 공유하고, 암/복호 모드에 따라서 XOR 연산의 입력을 달리 하는 멀티플렉서(MUX) 를 사용하여 암호화 키와 복호화 키를 모두 생성할 수 있는 키 생성 모듈을 구현하였다. 그림3에 구현한 128 비트 키 생성 모듈을 나타내었다.

그림 3에서 알 수 있듯이 암호화용 라운드키는 32비트 단위로 순차적으로 생성되므로 키 생성에 총 4 클럭이 소요된다. 그러나 데이터의 암호화시 라운드키는 MixColumn 변환후 사용되고 MixColumn 변환에 4 클럭이 소요되기 때문에 라운드 연산의 수행과 라운드 키 생성을 병렬로 처리해도 타이밍에 문제가 없다. 또한 복호화용 라운드 키생성에는 두 클럭이 소요되고, 라운드 키는 라운드 연산이 시작된 후 두 클럭 후에 사용되므로 복호화 연산에서도 라운드 키의 on-the-fly 생성 방식은 라운드 연산에 영향을 미치지 않는다.

Ⅴ. Rijndael 암호 프로세서의 설계 및 합성결과

본 논문에서 설계한 Rijndael 암호 프로세서는 32 비트 버스를 통해서 데이터와 키, IV 값이 들

어오면 암호 코어 내부 레지스터에 저장이 된다. 외부에서 Rijndael 블록 암호의 다섯가지 모드 (ECB 모드, CBC 모드, OFB 모드, CFB 모드, CTR 모드) 중 한 가지 모드를 선택하는 신호가 들어오면 각 모드에 따른 연산을 수행한 후 암호 코어의 입력 데이터으로 들어간다. 각 라운드 연산이 수행되고 마지막 라운드 연산의 결과가 나오면 done 신호가 '1' 이 되고 결과 데이터는 32비트 버스를 통해서 출력된다.

라운드 연산은 64비트 단위로 수행되는데, 먼저 Shift_Row(or Inverse Shift_Row)변환을 한 후 상위 64비트에 대해서 8개의 S-box(or SI-box)를 이용하여 Substitution(or Inverse Substitution) 변환을 수행한다. 이 결과값을 이용하여 MixColumn변환과 Add_Round_Key 변환 (Add_Round_Key변환과, Inverse MixColumn 변환)을 수행한다.

설계한 프로세서를 0.25 um CMOS 공정을 사용하여 50MHz에서 synopsys로 합성한 결과 64 비트 단위로 라운드 연산을 수행할 때 128 비트 단위로 라운드 연산을 수행하는 것 보다 34%의 게이트 수 감소를 얻을 수 있음을 알 수 있었다. 또한 최악 지연시간이 0.32 ns 이므로 최대 동작 주파수는 대략 3GHz 이고, ECB 모드의 경우 암/복호율은 약 2.56Gbps 이다.

Ⅵ. 결론

Rijndael 알고리듬을 하드웨어로 구현함으로써 많은 데이터를 빠른 시간안에 암/복호화 할 수 있다. 64비트 단위의 라운드 연산과 S-box/SI-box의 구현 방법, 라운드 키 생성 방법, MixColumn 용 다항식 곱셈기의 구현 등을 통해서 대략 18000 게이트의 크기가 나왔으며, 최대 지연 시간을 고려했을 때 3GHz를 넘어서는 주파수에서 동작 가능할 것으로 보인다. 또한 이를 고려한 데이터 암/복호율은 2.56 Gbps로 아주 높은 성능을 보였다. 따라서 본 논문에서 소개한 Rijndael 프로세서는 smart card, SIM card, PDA, 휴대폰 등 저면적, 저전력을 요구하는 모바일 시스템에 적용하기에 적절할 것으로 보인다.

참고문헌

[1] http://csrc.nist.gov/encryption/aes/

[2] Elbirt, W Yip, B Chetwynd, C Paar, "An FPGA-BASED Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists," IEEE Trans. On VLSI

[3] Y.K.Lee, Y.S. Park, S.I.Jeon, "Rijndael S-box의 세 가지 구현 방법에 따른 FPGA 설계," 전자공학회 하계종합 학술대회, 제 25권, 제 1호, pp. 281~284, June, 2002.

[4] J. Wolkerstorfer, E. Oswald and M. Lamberger, "An ASIC Implementation of the AES SBoxes," CT-RSA 2002, LNCS 2271, pp. 67-78, 2002.

[5] A. Menezes, P.Van Oorschot, and S. Vanstone, "Handbook of Applied Cryptography," CRC Press, New York, 1997.