디지털 서명을 위한 고속 RSA 암호 시스템의 설계 및 FPGA 구현※

강 민 섭† · 김 동 욱††

요 약

본 논문에서는 기존의 Montgomery 알고리듬을 개선한 고속 모듈러 곱셈 알고리듬을 제안하고, 이를 기본으로 하여 디지털 서명에 적용 가능한 1024비트 RSA 암호 시스템의 설계 및 구현에 관하여 기술한다. 제안된 방법은 부분합 계산시 단지 1번의 덧셈 연산이 필요하지만, 기존 Montgomery 알고리듬에서는 2번의 덧셈연산이 요구되므로 기존 방법에 비해 계산 속도가 빠르며, 하드웨어 면적도 매우 감소된다.

제안된 RSA 암호 시스템은 VHDL(VHSIC Hardware Description Language)을 이용하여 모델링하였고, Synopsys™사의 Design Analyzer를 이용하여 논리합성(Altera 10K lib. 이용)을 수행하였다. 또한, FPGA 구현을 위하여 Altera MAX+ PLUS II 상에서 타이밍 시뮬레이션을 수행하였다. 실험을 통하여 제안된 방법은 계산 속도가 매우 빠르며, 하드웨어 면적도 매우 감소함을 확인하였다.

키워드 : 모듈러 곱셈, 몽고메리 멱승 알고리듬, RSA 암호 시스템, 디지털서명, VHDL 모델링, FPGA 설계, 해쉬 코드

Design and FPGA Implementation of a High-Speed RSA Algorithm for Digital Signature

Min-Sup Kang† · Dong-Wook Kim††

ABSTRACT

In this paper, we propose a high-speed modular multiplication algorithm which revises conventional Montgomery's algorithm. A hardware architecture is also presented to implement 1024-bit RSA cryptosystem for digital signature based on the proposed algorithm. Each iteration in our approach requires only one addition operation for two n-bit integers, while that in Montgomery's requires two addition operations for three n-bit integers. The system which is modelled in VHDL(VHSIC Hardware Description Language) is simulated in functionally through the use of Synopsys™ tools on a Axil-320 workstation, where Altera 10K libraries are used for logic synthesis. For FPGA implementation, timing simulation is also performed through the use of Altera MAX+ PLUS II. Experimental results show that the proposed RSA cryptosystem has distinctive features that not only computation speed is faster but also hardware area is drastically reduced compared to conventional approach.

Key word : Modular multiplication, Montgomery's exponentiation algorithm, RSA cryptosystem, Digital signature, VHDL modeling, FPGA design, Hash code

1. 서 론

최근 통신 기술의 발전으로 컴퓨터 통신망을 통한 정보 교환이 활발히 이루어지면서 컴퓨터 통신망의 불법적인 접근을 통한 정보 노출이 용이하기 때문에 암호 알고리듬에 대한 연구가 활발히 진행 중에 있다[1, 2]. 암호 알고리듬은 키 사용 방법에 따라 공통키(common key encryption) 암호 방식과 공개키(public key encryption) 암호 방식으로 나눌 수 있다. 공통키 암호화(비밀키) 방식은 암·복호화 과정에서 동일한 키를 이용하는 방식으로 대표적인 알고리듬으로 DES (Data Encryption Standard)[1]가 있으며, 공개키 암호화 방식[2]은 공통키 방식에서 문제가 되었던 키의 분배 문제와 인증 문제를 해결하기 위하여 암호화 과정과 복호화 과정에서 서로 다른 두 개의 키, 즉 공개키(public key)와 비밀키(private key)를 이용한다. 이 방법은 침해자가 암호화 키를 알아 내도 암호화 키로부터 복호화 키를 만들어 낼 수 없다는 특징을 가지고 있다. 대표적인 알고리듬은 RSA (Rivest, Shamir, Adleman)이며[2], 이 알고리듬은 디지털 서명(digital signature)에도 사용 가능하다는 측면에서 그 응용 범위가 매우 넓다.

1973년에 Shannon의 Confusion과 Diffusion에 기반을 둔 알고리듬인 Lucifer[3]가 발표된 이래, 미 상무성의 국립 표준국(NBS) 주관으로 대칭 키 값를 기반으로 한 DES 암호 알고리듬[1]이 1977년에 발표되었다. 또한, 1976년에 Diffie와 Hellman에 의해 공개키 알고리듬[4]이 최초로 제안되었고, 그후 공개키 개념을 실현하기 위한 다양한 알고리듬이 발표되었지만, 현재 안정성을 인정받고 있는 알고리듬은 RSA와 Diffie-Hellman 알고리듬이다. RSA 알고리듬 ($M = C^d \bmod N$)은 모듈러 N에 대한 소인수 분해의 어려움에

※ 이 논문은 (1998)년 한국학술진흥재단의 학술연구비에 의하여 지원되었음.
† 종신회원 : 안양대학교 정보통신컴퓨터공학부 교수
†† 준 회 원 : 건국대학교 대학원 컴퓨터공학과
논문접수 : 2000년 5월 30일, 심사완료 : 2000년 8월 21일

그 안정성을 두고 있으며, 공개하는 키 중 하나인 모듈러 N 정수의 소인수 분해는 NP-문제로 알려져 있다[2].

RSA 암·복호 알고리듬은 모듈러 멱승 연산을 기본 연산으로 사용하고 있고, 이 모듈러 멱승 연산은 모듈러 곱셈 연산을 원자적 연산으로 하여 수행된다. 기존의 RSA 알고리듬을 이용할 경우 사용하는 키 값이 증가하면 모듈러 멱승을 위한 연산 속도가 감소되므로 고속연산을 위해서는 멱승에 필요한 모듈러 곱셈 수를 가능한 한 최소한으로 줄여야 한다.

따라서 공개키 암호 알고리듬의 실시간 처리를 위해서는 알고리듬 레벨에서의 고속화 기법 연구와 더불어 그 알고리듬의 구현 방법에 대한 연구도 매우 중요하다[5-10].

본 논문에서는 기존의 Montgomery 알고리듬[11]을 개선한 고속 모듈러 곱셈 알고리듬을 제안하고, 이를 기본으로 하여 디지털 서명에 적용 가능한 1024비트 RSA 암호 시스템의 설계 및 FPGA 구현에 관하여 기술한다. 제안된 방법은 부분합 계산시 단지 1번의 덧셈 연산이 필요하지만, 기존 Montgomery 알고리듬에서는 2번의 덧셈연산이 요구되므로 기존 방법에 비해 계산 속도가 빠르며, 하드웨어 면적도 매우 감소된다. 제안된 알고리듬은 VHDL(VHSIC Hardware Description Language)을 이용하여 설계하였고, Synopsys™사의 Design Analyzer를 이용하여 논리합성(Altera 10K 라이브러리 이용)을 수행하였다. FPGA 구현을 위하여 Altera MAX+ PLUS II 상에서 타이밍 시뮬레이션을 수행하였고, 실험을 통하여 제안된 방법은 계산 속도가 매우 빠르며, 하드웨어 면적도 매우 감소함을 확인하였다.

본 논문은 다음과 같이 구성되어 있다. 제 2장에서는 디지털 서명을 위한 RSA 암호 시스템에 대한 관련 연구에 대해서 설명한다. 제 3장에서는 RSA 암호 시스템의 고속화를 위한 개선된 모듈러 곱셈 및 멱승 알고리듬을 제안한다. 제 4장에서는 VHDL을 이용한 제안된 고속 RSA 시스템의 하드웨어 설계를 기술하고, 제 5장에서는 구현 결과 및 성능 분석 결과를 기술한다. 제 6장에서는 결론을 내린다.

2. 관련 연구

2.1 RSA 암호 시스템

RSA 암호 시스템에서 암·복호화를 위한 개인키와 공개키는 다음의 절차에 의해서 구할 수 있다[2].

[1] 두 개의 커다란 솟수 P와 Q를 선택하여 N을 계산한다.

$$N = P \times Q$$

[2] (P-1)×(Q-1)과 서로소인 D를 선택한다.

$$\gcd((P-1)\times(Q-1), D) = 1$$

[3] modular(P-1)×(Q-1)에 대한 D의 곱셈의 역원(multiplication inverse)을 구한다.

$$E \times D \equiv 1 (\mathrm{mod}(P-1)\times(Q-1))$$

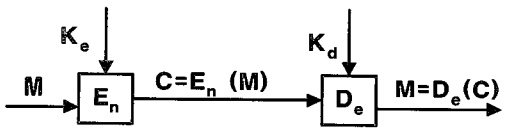
[4] (E,N)는 개인키, (D,N)은 공개키로 정의하며, (D, N)은 공개하고 E는 비밀리에 보관한다.

RSA 암호 시스템에서 암호화와 복호화는 각각 식 (1)과 (2)와 같이 정의된다.

$$C = En(M) \equiv M^E \pmod{N} \quad (1)$$

$$M = De(C) = C^D \pmod{N} \quad (2)$$

여기서 *En*과 *De*는 각각 Encryption과 Decryption을, M과 C는 각각 평문과 암호문을 나타낸다.



(그림 1) RSA 공개키 암호 시스템의 원리

(그림 1)은 RSA 암호 시스템의 원리를 나타낸다. 수신자는 키 분배 센터에 자신의 공개키 K_e를 등록하여 공개하고, K_d를 비밀리에 보관한다. 그리고, 송신자는 키 분배 센터에 등록된 수신자의 공개키 K_e를 얻어 수신자와 통신을 수행할 수 있다.

송신자는 메시지 M를 수신자의 공개키 K_e와 RSA 암호 알고리즘 E_n를 이용하여 암호화하여 비문 C를 생성한 다음 수신자에게 보낸다. 수신자는 자신의 비밀키 K_d와 RSA 복호 알고리즘 D_e를 이용하여 암호문 C를 복호화하여 메시지 M을 구한다. 이때 키 K_e와 K_d는 서로 다르며 공개된 키 K_e로부터 비밀키 K_d를 구하기는 어려워야 한다. 이와 같이 A에게 비밀 통신을 하고자하는 사람은 누구라도 A의 공개키를 가지고 송신할 내용을 암호화하여 A에게 전송하면, A는 자신만이 가지고 있는 비밀키를 이용하여 암호문을 평문으로 복호화할 수 있다. RSA와 같은 공개키 암호화 시스템에서는 공개키를 이용하여 상대방에게 보낼 메시지를 암호화하기 때문에 이 메시지가 위조될 수 있는 위험성이 존재할 수 있다. 이러한 문제점을 해결하기 위해 송신자의 신원을 확인할 수 있는 방법, 즉 디지털 서명(digital signature)이 필요하다[10].

2.2 RSA 시스템을 이용한 디지털 서명

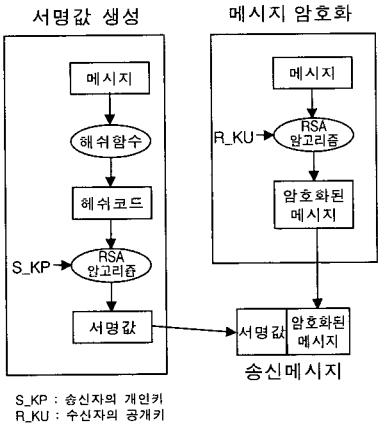
디지털 서명이란 상대방에게 전송된 메시지에 대해 그 내용이 수정이나 위조되지 않았음을 보장하는 동시에 메시지의 주체인 사용자들이 정확함을 제3자가 확인할 수 있게 해주는 인증 방식이다. 디지털 서명의 구현 방안은 여러 가

지가 있으나 RSA 암호 시스템을 이용한 방식은 기존 공개키 암호 알고리듬 중에서 높은 안정성이 인정되고 있다[9, 10]. 디지털 서명을 위해 송신측에서 증거 생성을 하고 수신측에서 검증을 하기 위한 2가지 과정이 필요하다.

2.2.1 증거 생성 과정

증거를 생성하기 위해서는 디지털 서명을 위한 메카니즘 즉, 메시지의 크기를 줄이기 위한 해쉬 함수(hash fuction)와 암호화를 위한 공개키 알고리즘이 필요하며, 본 연구에서는 RSA 알고리듬을 이용한다.

(그림 2)는 기밀성 서비스가 요구되었을 경우 디지털 서명을 위한 증거 생성 과정을 나타낸다.



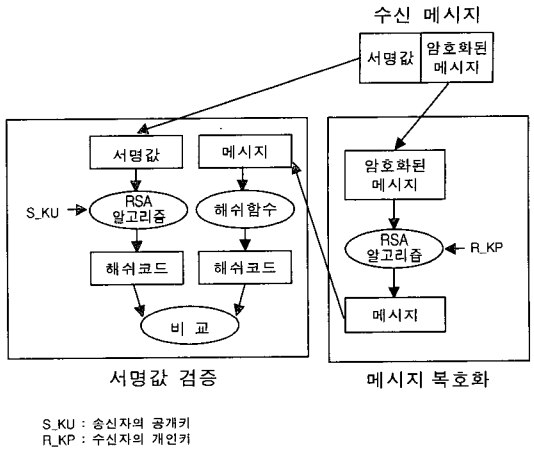
(그림 2) 디지털 서명을 위한 증거 생성 과정

먼저, 서명값 생성을 위해 송신자는 보낼 메시지를 해쉬 함수를 이용하여 해쉬 코드를 생성한 후, RSA 알고리듬과 송신자의 개인키(S_KP)를 이용하여 메시지에 대한 서명값을 구한다. 다음에 RSA 알고리듬과 수신자의 공개키(R_KU)를 이용하여 메시지를 암호화하여 서명값과 암호화된 메시지를 송신측에 보낸다.

2.2.2 증거 검증 과정

(그림 3)은 기밀성 서비스가 요구되었을 경우 디지털 서명을 위한 증거 검증 과정을 나타낸다. 수신측에서는 송신측으로부터 수신된 메시지에 대한 수정, 변조 등의 유무를 확인해야 한다.

먼저, 수신자의 개인키(R_KP)를 가지고 암호화된 메시지를 RSA 알고리즘을 통하여 복호화한다. 그리고 복호된 메시지를 해쉬 함수를 이용하여 해쉬 코드(B)를 구한다. 한편, 수신한 서명값을 송신자의 공개키(S_KU)를 가지고 RSA 알고리즘을 통하여 해쉬 코드(A)를 구한다. 마지막으로 해쉬 코드 A와 B를 비교하여 이 값이 서로 같으면 수신된 메시지가 정상적인 것이고, 만약 다를 경우에는 메시지에 이상이 발생한 것이므로 에러로 처리하여 제어를 넘긴다.



(그림 3) 디지털 서명을 위한 검증 과정

이때 수신측에서는 증거물로서 수신된 서명값과 메시지를 함께 보관한다. 이렇게 생성된 서명은 오직 송신자만이 만들 수 있으므로 수신자는 서명자를 인증할 수 있다. 또한 허가된 수신자만이 복호화할 수 있으므로 메시지의 기밀성을 유지할 수 있다.

2.3 모듈러 멱승

앞에서 설명한 RSA 암호 알고리듬의 고속화에 있어서 모듈러 멱승 연산은 매우 중요하다. 모듈러 멱승을 위한 연산은 수 많은 모듈러 곱셈에 의하여 수행된다. 따라서 고속 RSA 암호 시스템을 구현하기 위해서는 커다란 승수를 갖는 모듈러 곱셈 연산에 대한 고속 연산 알고리듬의 설계가 매우 중요한 요소로 작용한다.

2.3.1 Montgomery 알고리듬

가장 널리 사용되는 모듈러 감소 알고리듬의 하나인 Montgomery 알고리듬[11]은 수 체계의 변환을 통해 모듈러 감소가 쉽게 되는 수 체계에서 연산한 후 원래의 수체계로 역변환시키는 방법을 이용한다. 즉, 모듈러 감소가 쉬운 수 $R = 2^n$를 이용하여 모듈러 곱셈을 비교적 빠르게 할 수 있는 알고리듬이다. 일반적으로 Montgomery 알고리듬은 RSA 암·복호 알고리듬의 H/W구현에 가장 널리 사용되는 알고리듬으로 알려져 있다[6-7].

Montgomery 알고리듬은 $MonPro(A, B) = A \cdot B \cdot r^{-1} \bmod N$을 계산한다. 여기서 N은 솟수이며, $0 \le A$ and $B < N$이라 가정하고, r은 N과 서로 소인 N 보다 큰 수이다. 그리고 $A = \sum_{j=0}^{n-1} a_j \times 2^j$라고 하면 Montgomery 알고리듬을 이용하여 부분합인 S[0], S[1], …, S[n]을 구할 수 있다. (그림 4)에 보인 함수 MonPro()는 C. D. Water에 의해서 하드웨어 구현에 적합하도록 변형된 Montgomery 알고리듬을 나타낸다.

(그림 4)의 알고리듬에서 S[0] = 0으로 초기화되며, for 루프는 i의 값에 의해서 제어된다. 이러한 결과는 "$2^n S = A \cdot B$

+M·N"을 만족시키며 "$S = A \cdot B \cdot 2^{-n} \bmod N$"이 된다. 이 때 S는 "S<2N"이 되므로 출력값은 S+N 또는 S가 된다.

```
MonPro(A, B, N)
  S[0] := 0;
   for i in 0 to n-1
    q := (Si + Ai×B) mod 2;
    S[i+1] := (S[i] + Ai×B + q×N) div 2;
  return S[i+1];
```

(그림 4) Montgomery 알고리듬

2.3.2 Montgomery 멱승 알고리듬

Binary method[1]에 의해 $m^e \bmod n$을 구현할 때, Montgomery product를 사용할 수 있다. 이는 지수 비트를 왼쪽에서 오른쪽으로 또는 그 반대로 스캔하여 각 단계마다 제곱이 발생하며, 스캔된 비트에 따라 선택적인 곱셈이 발생하게 되는데 (그림 5)는 Montgomery 멱승(exponentiation) 알고리듬을 나타낸다[11].

```
MonExp(M, N, E)
  M := M · R mod N;
  X := R mod N;
  for i in k - 1 downto 0
      X := MonPro(X, X);
      if Ei = '1' then
         X := MonPro(M, X);
  X := MonPro(X, 1);
  return X;
```

(그림 5) Montgomery 멱승 알고리듬

MonExp()는 MonPro()의 계산 결과 값의 특성 때문에 M과 X를 계산하여 for 루프 문에서 사용하여야만 올바른 결과 값을 얻을 수 있다. 이 알고리듬의 문제점은 for 루프 안에서 연산이 수행될 때 매우 큰 계산량이 요구된다는 것이다.

3. 고속 모듈러 곱셈 및 멱승 알고리듬

3.1 고속 모듈러 곱셈 알고리듬

(그림 4)에 나타난 기존의 알고리듬[7]은 키 값이 커지면 커질수록 반복문 내에서 처리를 위해 커다란 연산량이 요구되므로 실시간 처리에 문제점을 가지고 있다. 이러한 문제점을 해결하기 위해서는 가장 커다란 연산량이 되는 덧셈 연산의 횟수를 감소시켜야 한다. (그림 6)에 나타난 함수 AM()은 기존의 Montgomery 알고리듬을 개선한 제안된 고속 모듈러 곱셈 알고리듬을 나타낸다.

기존 알고리듬(그림 4)에 있어서 S[i+1]의 연산은 for 루프에서 두 번의 덧셈 연산이 요구된다. 그러나 제안된 알고리듬(그림 6)은 고정된 값으로 A_i와 q에 의해서 덧셈 연산의 피연산자(operand)로 사용되어지기 때문에 단지 한번의 덧셈 연산만이 필요하다. 즉, 입력 값인 B, N과 0 값은 특별한 연산 및 기억 공간을 필요로 하지 않고, 단지 for 루프 문에 들어가기 전에 B+N값을 한번 계산하여 특정의 기억 공간에 저장시키고, A_i와 q에 의해서 선택된 값으로 S[i]와 덧셈 연산을 수행하면 된다. 결과적으로 단지 한 번의 덧셈 연산이 요구되므로 모듈러 곱셈의 고속화가 가능하게 된다.

```
AM(A, B, N)
   S[0] := 0;
   BN := B + N;
   for i in 0 to n-1
       q := (S[i] + Ai×B) mod 2;
       case(Ai & q) of
           "11" : opd := BN;
           "10" : opd := B;
           "01" : opd := N;
           "00" : opd := 0;
        end
        S[i+1] := (S[i] + opd) div 2;
   return S[i+1];
```

(그림 6) 제안된 고속 모듈러 곱셈 알고리듬

3.2 고속 모듈러 멱승 알고리듬

기존의 MonExp() 함수(그림 5)에서는 모듈러 멱승 연산에 앞서 피연산자에 대하여 수 체계 변환 작업을 수행하기 위하여 추가적인 연산이 요구된다. 따라서 하드웨어로 구현시 수 체계 변환 연산을 수행하기 위하여 추가적인 연산 블럭을 구현하여야 하는 문제점을 갖게 된다. 또한 기존 모듈러 멱승 알고리듬은 for 루프 안에서 발생하는 모듈러 제곱과 모듈러 곱셈 연산으로 인하여 커다란 계산량이 요구된다. 이러한 문제점을 해결하기 위해서 본 논문에서는 새로운 2개의 변수 즉, $R4(2^{2n+2})$와 $R2(2^{n+1})$를 정의하여 개선된 고속 모듈러 멱승 알고리듬을 제안한다. (그림 7)은 제안된 알고리듬인 함수 AdvancedMonExp()를 나타낸다.

```
AdvancedMonExp( M, N, E )
   M := AM(R4, Mess)
   X := R2
   for (i = n-1; i >= 0; n--)
      X := AM(X, X);
      if Ei = '1' then
        X := AM(M, X);
        end if;
   return AM( X, 1 );
```

(그림 7) 제안된 고속 모듈러 멱승 알고리듬

기존 알고리듬[11]에서의 수체계 변환을 위한 "M := M·R mod N"의 계산은 입력값이 "0 ≤ A and B < N" 조

건을 만족시킬 수 없다. 따라서 Monpro()를 사용하여 연산을 수행할 수 없고, 이 계산을 위한 새로운 연산 과정이 필요하다. 이러한 과정의 추가는 모듈러 멱승 연산 과정이 복잡할 뿐만 아니라, 하나의 키에 대해서 연속적으로 여러 개의 블록 메시지를 암·복호화 하는 경우 계산량이 증가하게 된다[12].

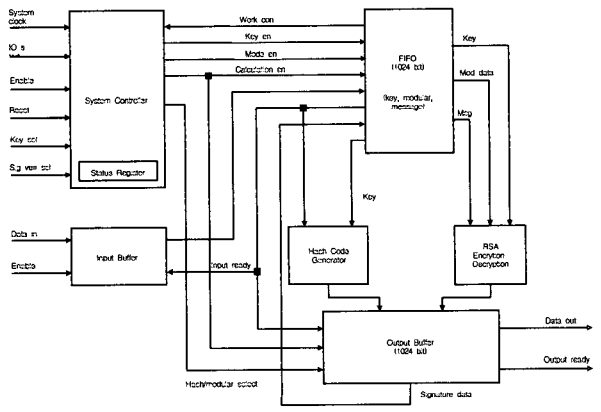
이러한 문제점을 해결하기 위해서 제안된 방법에서는 R4와 R2를 정의하고, "$R4(2^{2n+2})$ mod N"과 "$R2\ (2^{n+1})$ mod N" 연산을 수행한다. 여기서 상수 R4와 R2는 모듈러 N에 의해서 생성된 값으로서, 모듈러 N이 변화되지 않는 한 이 두 상수는 변화하지 않는다. 결과적으로 R4와 R2를 사용하면 조건 "$0 \le A$ and $B < N$"이 만족하므로 추가적인 연산 과정 없이 모듈러 멱승 연산이 간단하며, 계산량이 감소하게 된다.

한편, 기존 Montgomery 알고리듬의 결과값은 "$2^nS = A \cdot B + M \cdot N$"이 되므로 "$S = A \cdot B \cdot 2^{-n}$ mod N"이 된다. 따라서 결과 값 S는 "$S < 2N$"이 되므로 이 결과를 바로 다음 계산에 사용할 수 없고, S 값이 "$S + N$" 경우를 체크하여 N을 빼주어야만 한다.

제안된 모듈러 멱승 알고리듬에서는 기존 알고리듬에서 사용하는 $R = 2^n$ 대신에 $R2 = 2^{n+1}$를 모듈러 곱셈 알고리듬의 입력값으로 사용하여 문제점을 해결하였다.

4. 고속 RSA알고리듬의 하드웨어 설계

이 장에서는 제안된 고속 모듈러 멱승 알고리듬을 기본으로 하여 디지털 서명을 위한 RSA 암호 시스템의 설계를 기술한다. (그림 8)은 제안된 RSA 암호 시스템의 전체 블록도를 나타낸다. 이 시스템은 크게 6개의 모듈, 즉 데이터 입력부(Input Buffer), 데이터 생성부(FIFO), 해쉬코드 생성부(Hash_Code Generator), RSA 암·복호 연산부(RSA Encryption_Decryption), 데이터 출력부(Output_Buffer), 그리고 시스템 제어부(System Controller)로 나눌 수 있다.



(그림 8) 제안된 RSA 암호 시스템의 전체 블록도

4.1 데이터 입력부

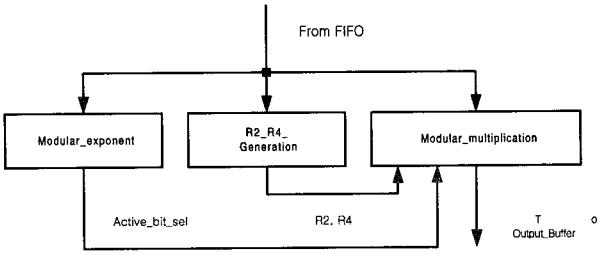
데이터 입력부는 Data_in을 통하여 서명값 생성 또는 검증을 위한 메시지 M과 RSA 암·복호화를 위한 3개의 키 값을 32비트 단위로 입력받아 임시 저장하는 레지스터이다. 레지스터에 임시 저장되어진 값들은 FIFO(First In First Out)로 전달되어, 1024비트의 데이터를 구성한후, RSA 암·복호와 해쉬 연산에 이용된다.

4.2 데이터 생성부

데이터 생성부는 1024비트의 데이터를 저장하기 위한 FIFO 레지스터로 구성된다. 또한 내부의 메시지와 키 값의 흐름을 제어한다. 보관되어지는 키 값은 크게 3가지의 종류로 구분되는데, 이들은 각각 인증기관으로부터 네트웍으로 접촉하고자 하는 사람의 인증된 공개키를 받기 위해 사용하는 master key, 자신의 메시지를 암호화 하여 상대에게 보내고자 할 때 사용하는 encrypt key, 그리고 암호화 되어 자신에게 들어온 메시지를 복호화하기 위하여 사용하는 decrypt key이다. 또한 구분된 각각의 키는 지수승 E와 모듈러 N으로 구성되어 각 키마다 2048비트의 키 저장 공간을 가지며, 메시지는 1024비트의 저장공간을 갖는다.

4.3 RSA 암·복호 연산부

생성부에서부터 1024비트 메시지와 키 값(E,N 또는 D,N)을 받아 RSA 암·복호 연산부는 RSA 암호 시스템에서 실제로 암복화와 복호화를 수행하는 가장 핵심적인 블록으로 3개의 모듈로 구성되어 있다(그림 9).



(그림 9) RSA 암·복호 연산부의 내부 블록도

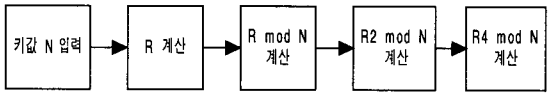
4.3.1 Modular_exponent 블록

RSA 암·복호화 연산인 모듈러 멱승 연산시 1024비트인 지수승 키 E를 외부 클럭에 맞추어 MSB에서부터 한 비트씩 스캔하고 각 비트의 값에 따라 Modular_multiplication (M_n) 블록의 입력값을 제어하여 모듈러 멱승 계산을 수행하는 역할을 수행한다.

4.3.2 R2_R4_Generation 블록

이 블록에서는 제안된 고속 암·복호 알고리듬을 실행하기 위하여 $R4(2^{2n+2})$와 $R2(2^{n+1})$을 계산하여 M_m 블록의 입

력값으로 사용도록 한다. 따라서 이 블록은 계산을 위한 초기값을 구하는 부분이므로 가장 먼저 데이터를 받아 연산을 시작하며, 이 블록의 동작이 완료되었을 때 모듈러 멱승 연산이 수행될 수 있다. 또한 데이터 생성부로부터 키 N을 받아 임시 저장하고 M_m의 입력값으로 보내는 역할도 수행한다. (그림 10)은 R2와 R4를 생성하는 과정을 나타낸다.

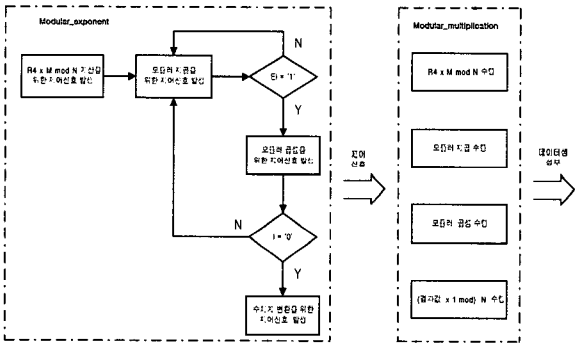


(그림 10) R2, R4 생성과정

4.3.3 Modular_multiplication 블록

제안된 고속 모듈러 곱셈 알고리듬을 사용하여 모듈러 곱셈을 구현하기 위하여 개선된 알고리듬에 필요한 adder와 systolic array가 기본 블록으로 구성되어 있다. 하드웨어 면적 문제를 해결하기 위하여 외부 클럭에 맞추어 출력값이 순환적으로 입력되는 계산 형태를 취하고 있다.

위에서 설명한 M_m과 M_e(Modular_exponent) 블록 간의 모듈러 멱승을 위한 처리 과정은 (그림 11)과 같다.



(그림 11) M_e 블록과 M_m 블록의 동작

4.4 해쉬 코드 생성부

해쉬 코드 생성부는 서명값 생성 및 검증을 위한 해쉬 코드를 생성하기 위한 블록이다. 본 논문에서는 해쉬 코드를 생성하기 위하여 NIST에서 개발된 SHA을 기본으로 하고 있다[15]. 이 알고리듬은 입력된 2^{64} 비트 미만의 길이를 갖는 메시지를 512비트 블록 단위로 2회 처리하여 160비트 메시지를 출력한다. 하나의 512비트의 블록을 처리하기 위해서는 80 라운드가 소요된다.

4.5 데이터 출력부

데이터 출력부는 2 종류의 데이터를 저장하는 역할을 한다. 즉, RSA 암·복호부에서 모듈러 멱승 연산 결과(1024비트의 암·복호 데이터) 또는 디지털 서명에서 증거 생성 또는 검증을 위한 해쉬 코드(512×2비트)를 임시 저장한다. 저장된 값은 다음 연산을 위해 재 사용되어질 수 있고 또는 외부로 32비트 단위로 출력시킨다.

4.6 시스템 제어부

시스템 제어부는 시스템의 모든 입출력 및 연산 동작을 제어하기 위한 블록이다. 제어 입력 Sig_veri_sel은 외부로부터 서명값 생성 또는 검증을 위해 해쉬 코드 생성부와 RSA 암·복호 연산부를 제어하기 위한 2비트의 신호선이다. 즉, 데이터의 흐름을 세부적으로 구분하면 제어신호에 따라 (1) 해쉬 코드 생성부의 동작, (2) RSA 암·복호 연산부 동작, (3) RSA 암·복호 연산부 동작 후 그 결과값에 대한 해쉬 코드 생성부의 동작, (4) 해쉬 코드 생성부의 결과값에 대한 RSA 암·복호 연산부의 동작이 이루어진다.

Key_sel은 RSA 암·복호 연산부에서 사용 가능한 3개의 키 값 즉, 송신자의 개인키(S_KP)와 인증 기관의 공개키 또는 수신자의 공개키(R_KU)를 선택하기 위한 2비트의 신호선이다. 그리고, RSA 암·복호 연산부, 해쉬 코드 생성부, 그리고 데이터 생성부의 동작 여부는 시스템 내부의 status_register 상태에 따라 결정된다.

5. 구현 결과 및 성능 분석

본 논문에서 제안된 RSA 암복호 시스템은 크게 6개의 기능 모듈로 구성되어 있으며 Axil-320 워크스테이션 상에서 VHDL (VHSIC Hardware Description Language)을 이용하여 설계하였다. 제안된 1024비트 RSA 암호 시스템의 Front-end 설계는 Synopsys™사의 Design Analyzer를 이용하여 시뮬레이션 및 논리 합성을 수행하였고, 논리 합성 시 Altera 10K 라이브러리를 이용하였다. FPGA 구현을 위하여 Altera MAX+ PLUS II 상에서 타이밍 시뮬레이션을 수행하였다.

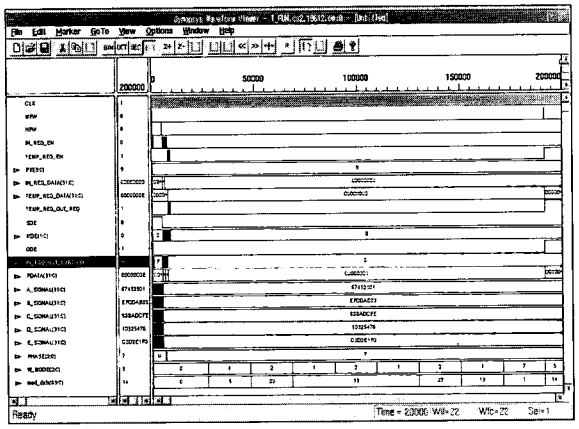
5.1 구현 결과

서명값 생성을 위해서는 RSA 암복호 알고리듬에서 2개의 입력, 즉 메시지에 대한 해쉬코드와 송신자의 개인키가 요구된다. (그림 12)는 디지털 서명에 제안한 방식을 적용하기 위해 서명값 생성을 위한 기능 레벨 시뮬레이션을 나타낸다. 초기에 1024비트 메시지가 32비트 단위로 입력 IN_REG_DATA를 통하여 해쉬 블럭으로 입력된다. 서명값 생성을 위해 사용된 메시지 M은 7이고, RSA 암복호화를 위해 사용된 개인키(E,N)은 (7,33)이다.

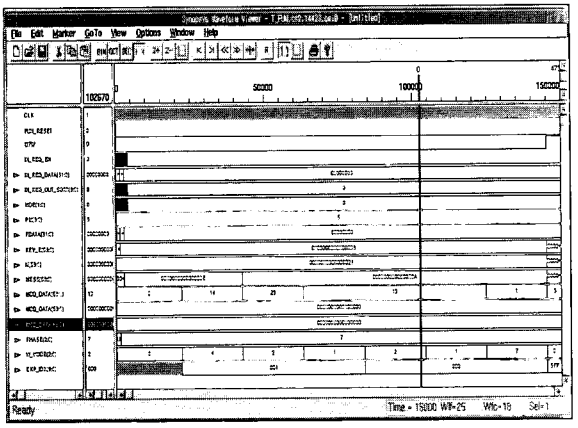
해쉬 함수에서는 메시지를 512비트 블록 단위로 2회 처리하여 160비트 메시지를 출력하게 되는데, 이때 주어진 M에 대한 출력 결과(해쉬 코드)는 DATA_ X_OUT에 나타나 있으며, 여기에서 X는 A, B, C, D, 그리고 E 이다. 그리고 RSA 암복호 연산을 위해 2.3 절에서 주어진 조건을 만족시키기 위하여 해쉬 코드는 비트 단위로 조작된다. 비

트 조작된 해쉬 코드(5)는 TEMP_REG_DATA를 RSA 암복호 블록으로 입력된다.

서명값을 생성하기 위해서는 해쉬 코드와 함께 송신자의 개인키(E,N)이 요구된다. 이에 대응되는 키 값은 결과의 IN_REG_DATA에 나타나 있으며, 32비트씩 RSA 암복호 블록에 입력된다. 이때 MOD_DATA에 나타난 시뮬레이션 결과가 생성된 서명값(E(14))이며, 이 값은 FIFO로 전송된다.



(그림 12) 서명값 생성에 대한 시뮬레이션 결과



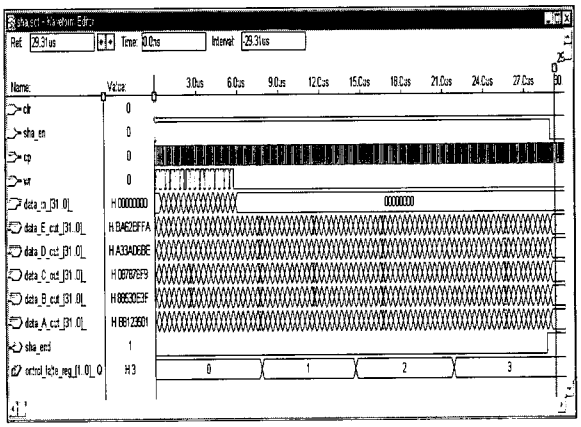
(그림 13) 서명값 검증에 대한 시뮬레이션 결과

검증 작업은 서명에 사용된 메시지를 입력으로 하여 계산된 해쉬 코드와 서명값과 송신자의 공개키를 입력으로한 RSA 복호화 결과값의 비교 작업이다. (그림 13)은 (그림 12)에서 생성된 서명값(14) 검증에 대한 기능 레벨 시뮬레이션 결과를 나타낸다.

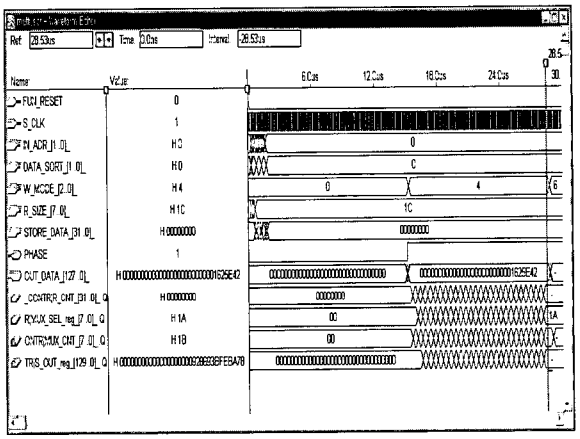
여기서 사용된 송신자의 공개키(D,N)는 (3,33) 이다. 시뮬레이션 결과에서 RSA 알고리듬의 출력 결과(MOD_DATA)는 5가 되므로 비교한 결과가 일치됨을 보여주고 있다.

(그림 14)는 키 값에 대한 Test_Vector를 구성한 후 Altera MAX+ PLUS II 상에서 FPGA 구현을 위한 해쉬 코드 생성부에 대하여 타이밍 시뮬레이션 결과를 나타낸다.

(그림 15)는 FPGA 구현을 위한 RSA 암·복호 연산부의 M_m에 대한 타이밍 시뮬레이션 결과를 나타낸다. 시뮬레이션 결과를 통하여 RSA 암·복호 연산 및 디지털 서명 및 검증이 정확하게 수행됨을 확인하였다.



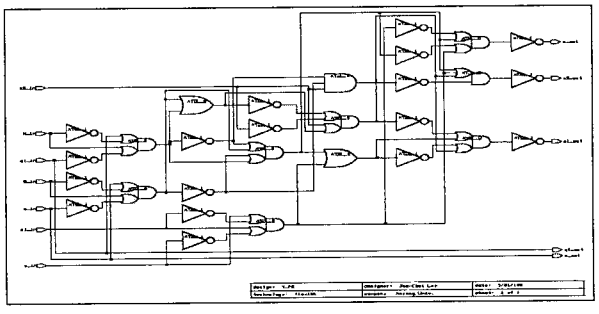
(그림 14) 해쉬 코드 생성부의 타이밍 시뮬레이션 결과



(그림 15) RSA 암·복호 연산부의 타이밍 시뮬레이션 결과

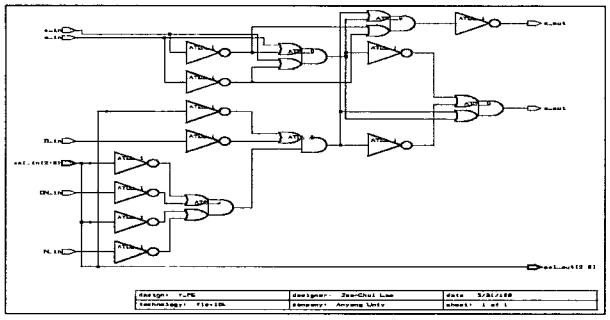
5.2 성능 분석

하드웨어 구현시 기존의 알고리듬[7]과 제안된 알고리듬의 성능을 비교하기 위하여 Synopsys™사의 Design Analyzer를 이용하여 논리 합성을 수행하였다. (그림 16)은 기존 알고리듬(그림 4)에 있어서 한 비트 모듈러 곱셈 연산을 위해 합성된 기본 셀을 나타낸다.



(그림 16) 기존 알고리듬의 기본 셀[7]

(그림 17)은 제안된 알고리듬(그림 6)에서 한 비트 모듈러 연산을 위해 합성된 기본 셀을 나타낸다.



(그림 17) 제안된 알고리듬의 기본 셀

제안된 시스템에서 모듈러 곱셈연산을 위한 블록은 64비트 단위로 처리되며, 이러한 블록들이 체인으로 구성되어 있다. <표 1>은 64비트 모듈러 곱셈 연산에 대해 기존의 방법[7]과 제안된 방법과의 성능 분석 결과를 나타낸다. 제안한 알고리듬의 유효성을 입증하기 위하여 기존의 방법[7]을 동일한 시스템상에서 VHDL을 이용하여 직접 구현한 후, 비교·분석 하였다. 항목(Items)에서 Used area는 Design analyzer를 이용하여 합성한 결과이며, Critical path는 Altera MAX+ PLUS II 상에서 Timing analyzer를 이용한 타이밍 결과를 나타낸다.

〈표 1〉 모듈러 곱셈 연산의 성능분석

Items \ Algorithms	Walter[7]	Proposed method
FA/primitive for basic cell	5/4	2/6
Used area(cell)	592	304
Critical path/Max delay(ns)	318.5	266.5

성능 분석 결과에서 임계 경로(critical path)인 Max delay는 약 16.3% 정도 개선되었고, 칩 면적은 약 48.6%가 감소되었다. 하드웨어 면적이 감소된 주된 이유는 모듈러 연산을 위해 제안된 방법에서는 Mux (multiplexer)와 FA를 사용한 기본 셀을 사용하기 때문이다. 즉, 기존의 방법에서는 FA와 기본소자(primitive gate)는 각각 5개, 4개로 구성되지만[7], 제안한 방법에서는 2개의 FA와 6개로 구성된 Mux를 사용하기 때문이다.

6. 결 론

본 논문에서는 기존의 Montgomery 알고리듬을 개선한 고속 모듈러 곱셈 알고리듬을 제안하고, 이를 기본으로 하여 디지털 서명에 적용 가능한 1024비트 RSA 암호 시스템의 구현에 관하여 기술하였다.

제안된 RSA 알고리듬은 Design Analyzer를 이용하여 논리합성을 수행하였다. 또한, 모듈러 연산 속도를 비교하기 위하여 MAX+ PLUS II 상에서 timing analyzer를 이용하여 시뮬레이션을 수행하였다. 시뮬레이션 결과를 통하여 서명값 생성 및 검증, 그리고 암복호화가 정확히 수행됨을 확인하였다.

FPGA로 구현된 암호 시스템은 기존 방법에 비해 계산 속도가 빠를 뿐만아니라, 하드웨어 면적도 매우 감소된다. 하드웨어 면적을 줄이기 위해서 모듈러 연산을 위한 기본 셀의 구성은 2개의 FA와 Mux를 사용하였다. 실험 결과를 통하여 연산속도는 기존의 방법에 비하여 약 16.3% 정도 개선되었고, 하드웨어 면적은 약 48.6%가 감소됨을 확인하였다.

참 고 문 헌

[1] NBS, 'Data Encryption Standard', FIPS Pub, 46, U.S, National Bureau of Standard, Washington DC, Jan. 1977.

[2] R. L. Rivest, A. Shrmir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-key Cryptosystems," Communications of the ACM, Vol.21, No.2, pp.120-126, Feb. 1978.

[3] A. Sorkin, "LUCIFER, A Cryptographic Algorithm," Cryptologia, Vol.8, No.1, pp.22-24, 1973.

[4] ANSI, "Data Encryption Algorithm," American National Standard X3, 92. NY. 1981.

[5] J. Sauerbrey, "A Modular Exponentiation Unit Based on Systolic Arrays," Proc. of AUSCRYPT '92, pp.12.19 -12.24, 1992.

[6] K. Iwamura, T. Matsumoto, and H. Imai, "Systolic Arrays for Modular Exponentiation using Montgomery Method" Proc. of Euro CRYPT '92, pp.477-481, 1992.

[7] C. D. Walter, "Systolic Modular Multiplication," IEEE Trans. on Computers, Vol.42, pp.376-378, 1993.

[8] 박태규, 황대준, "다중 프로세서를 위한 RSA 병렬 암호화 알고리즘 설계", 한국정보과학회 논문지, 제22권 제1호, pp.49-56, 1995.

[9] 강창구, 김대영, "디지틀 다중서명 방식 비교", 한국통신정보보호학회 학회지, 제2권, 제4호, pp.7-16, 1992.

[10] E. F. Brickell, "A Survey of Hardware Implementation of RSA," Proc. of CRYPTO89, pp.368-370, 1989.

[11] P. L. Montgomery, "Modular Multiplication without Trial Division." Mathema. of Computat., Vol.44, pp.519-521, 1985.

[12] 황효선, 임채훈, "공개키 암호 시스템의 고속 구현", 한국통신정보보호학회 종합학술발표회논문집, Vol.7, No.1, pp.232-247, 1997.

[13] J. Bobs and M. Coster, "Addition Chain Heuristics," Advances in Cryptology-CRYPTO'89, Proceedings, Lecture Notes in Computer Science, No.435, pp. 400-407, New York, NY : Springer-Verlag, 1989.

[14] C. K. Cok, High-Speed RSA Implementation, TR 201, November 1994.

[15] 최용락, 소우영, 이재광, 이임영, '통신망 정보 보호', 그린 출판사, 1996.

강 민 섭

e-mail : mskang@aycc.anyang.ac.kr

1979년 광운대학교 전자통신공학과 졸업 (공학사)

1984년 한양대학교 전자공학과 졸업 (공학석사)

1992년 일본 오사카대학교 전자공학과 졸업 (공학박사)

1984년~1993년 한국전자통신연구원 선임연구원

1986년~1987 일본 오사카대학교 연구원

1993년~현재 안양대학교 정보통신컴퓨터공학부 부교수

관심분야 : ASIC 설계, 암호보안, 무선 인터넷, 병렬처리 시스템

김 동 욱

e-mail : dwkim@cse.konkuk.ac.kr

1999년 안양대학교 컴퓨터학과 졸업 (공학사)

1999~현재 건국대학교 대학원 컴퓨터공학과 재학 중

관심분야 : 리눅스, 암호보안, 실시간 운영체제, PDA, 무선 인터넷