선형이동 Knapsack 공개키 암호시스템을 위한 프로세서 구현

正會員 白 寅 天*, 正會員 車 均 鉉**

The Implementation of Processor for Linearly Shift Knapsack Public Key Crypto System

In Cheon Paik*, Kyun Hyon Tchah* *Reguar Members*

요 약

선형이동 knapsack 공개키 암호를 위한 특수 프로세서의 설계를 보였다. 기존의 knapsack 보다 밀도를 높이고 벡터를 선형 이동시켜 비도가 증가된 선형이동 knapsack 시스템을 위한 구조를 구현하였다. 이 시스템의 성격상 각 경로에 따라 병렬 처리하는 것이 요구되어 이를 위한 파이프 라인식 병렬 구조를 제시하여 시스템을 VLSI로 구현 하였다. 또한 전체의 시스템의 성능을 평가하고 다른 시스템과 비교하였다. 시스템 성능은 디멘젼이 100인 경우 550kb/s의 속도로 데이터를 처리할 수 있다. 시스템 성능은 디멘젼이 100인 경우 550kb/s 의 속도로 데이터를 처리할 수 있다. 본 논문에서 제안한 암호와 시스템 구조를 확장하면 고속의 보안이 요구되는 곳에 이 시스템을 연결하여 사용할 수 있다.

ABSTRACT

This paper shows the implementation and design of special processor for linearly shift knapsack public key cryptography system. We highten the density of existing knapsack vector and shift the vectors linearly in order to implement the structure of linearly shift knapsack system which has the stronger cryptosystem. As it needs the parallel processing at each path according to the characteristics of this system, we propose the pipelined parallel structure and implement this system into VLSI.. Also we evaluate this system and compare with other systems.

The processing speed of this system is 550kb/s when dimension is 100. It is possible to use this system at the place of requiring high speed security to enlarge the structure of it.

* 순천향 대학교 전자통계학과
Dept. of Computer Science & Statistics, Soonchunhyang Univ.

** 고려대학교 전자공학과
Dept. of Electronics Eng., Korea Univ

論文番號 : 94216
接受日字 : 1994年 8月 10日

Ⅰ. 서 론

최근 암호화 시스템은 통신기술의 발달로 정보들의 교환이 늘어 나면서 보안의 필요성이 급격히 대두 되었고 컴퓨터 네트워크의 발달로 미지의 대상이 비밀 통신을 해야하는 이유로 공개키 시스템이 연구되고 있다.

대표적인 공개키 시스템으로 알려져 있는 것이 유한체에서 지수 계산에 근거한 P H(Pohlig Hellman)나 RSA(Rivest Shamir Adleman) Scheme 이다.[1] 또한 이러한 지수형 암호 시스템의 구현도 많은 연구가 진행되었다.[7][8][9][10] 다른 공개키 암호와 시스템으로 knapsack 시스템은 메모리 공간은 많이 소모하나 속도가 빠른 것으로 알려져 있다.[1] 기존의 knapsack 시스템은 정수 프로그래밍, diophantine 방식, short vector 알고리즘들에 의해 해독 되므로[2][3][4] 여러 개선 알고리즘들이 제시되었고 대표적인 알고리즘중의 하나가 고밀도 선형이동 knapsack 시스템으로 키 생성이 용이하고 다른 모듈라 곱으로는 얻을 수 없는 시스템이다. 선형이동 knapsack 시스템의 비도 실험에서 공격이 용이치 않음을 밝힌 바 있다.[5]

본 논문에서는 선형이동 knapsack 시스템을 구현하기 위해 제시된 알고리즘을 따라 이를 구현하기에 적절한 병렬 구조를 제안하고 VLSI 화를 위한 설계 및 결과를 설명한다. 전체를 추정 연산부, 복호화부, 암호화부, 결정부, 제어부의 5개의 개념적인 구성부분으로 나누었고 실제 설계에서는 메모리, 추정합계 연산(S″), 복호화, 암호화, 결정, 제어블럭으로 나누었다.[6]

전체 시스템 레벨에서 동작을 확인한 후 각 블럭들에 대해 논리 시뮬레이션을 수행하여 논리 동작을 확인하였다. 사용 공정은 ISRC 1.5μm double metal n-well CMOS 공정이고 Full Custom 방식으로 설계하였다.

2장에서는 기존의 knapsack 시스템과 고밀도 선형이동 암호화 시스템에 대하여 설명하고 3장에서는 제2장에서 제시된 시스템을 위한 병렬 구조를 보이고 4장에서는 VLSI 설계를 위한 신호 전략, 시뮬레이션 및 회로 구성을 설명한다. 5장에서는 시스템 성능을 분석하고 끝으로 결론을 보였다.

Ⅱ. Knapsack 암호화 시스템과 고밀도 선형이동 암호화 시스템

Merkle-Hellman이 제안한 knapsack암호 시스템은 최초로 송신자가

초 증가수열 $A' = (a_1', a_2', \cdots, a_n')$

(즉, $a_i' > \sum_{j=1}^{i-1} a_j'$)

를 선택하고 이것에 다음의 변형을 가하여 임의 수열 A=(a_1,a_2, …, a_n)을 만든다.[1]

$a_i = a_i' * W \bmod U$

여기서 $U > 2b_n$, gcd(W,U)=1로 U과 W를 선택한다.

따라서, A를 공개하고 송신자는 이진 메세지 M (m_1, m_2, … , m_n)에 $\sum_{i=1}^{n} m_i a_i$로 암호화 한다.

이제 송신자는 비밀 채널을 통하여 수신자에게 S를 보낸다. A가 공개되어 있고 도청자가 S를 가로채도 A의 부분합 S가 되는 것을 찾는 것은 NP complete문제이다.

하지만, 올바른 수신자, 즉 W와 U를 알고있으면 아래의 계산에 의해 메세지 M(m_1, m_2, …, m_n)을 얻을 수 있다.

$$\begin{aligned} S' &= W^{-1} S \bmod U \\ &= W^{-1} AM \bmod U \\ &= W^{-1} (WA')M \bmod U \\ &= A' M \bmod U \\ &= A' M \qquad\qquad W^{-1} = inv(W, U) \end{aligned}$$

초 증가 수열에서 m_i는 많아야 n번의 감산에 의해 계산될 수 있다. 이 시스템은 밀도가 0.5보다 작아 공개 화일 크기에 비효과적이고 복호키가 해독이 가능한 수 이라서 공격을 당하기 쉽다. 또한 Lenstra 등에 의해 제안된 L^3 reduction 알고리즘을 이용한 low density knapsack의 해법이 제시되었다. 이를 보완하기 위해 제시된 고밀도 선형 이동 knapsack 알고리즘을 설명한다. 이 고밀도 knapsack은 먼저 Merkle-Hellman의 저밀도 knapsack 을 고밀도 knapsack으로 바꾸고

다음에 이것을 다시 선형 이동시켜 새로운 벡터를 얻는 단계로 이루어진다.

먼저 기존 Merkle-Hellman의 저밀도 knapsack의 수열을 고밀도 knapsack으로 바꾸는 방법을 소개한다. 이것은 초증가 수열의 각 요소에서 다음 요소와의 차보다 작은 량을 그 요소에서 감해도 역시 초증가 수열을 이루는 성질 이용한 것이다. 아래의 절차에 의해 고밀도 knapsack으로 변환한다.[6]

단계 1 : 임의로 초 증가수열 A'=(a_1, a_2, …, a_n), gcd(W,U)=1인 W,U을 선택하고, v= ⌊W/U⌋ (여기서 ⌊x⌋는 floor function)

단계 2 : 암호화 키 $a_i = a_i' \times W \bmod U$ 계산

단계 3 : $h_i = a_i \bmod W$를 계산하여 고밀도 knapsack 을 생성한다.

단계 4 : $b_i = \lfloor a_i / W \rfloor$, $0 \le b_i \le v$, 를 계산하고, 복호화 키 $c_i = a_i - b_i$를 계산한다.

위의 절차에 의해 생성된 수열의 밀도는 0.94로 밀도가 0.5인 기존의 M H knapsack 에 비해 월등히 크다. 이 고밀도 knapsack 도 Shamir 나 Brickell, Adleman 과 Lagarias 의 알고리즘에 의해 해독될 수도 있다. 이를 위해 벡터의 각 요소들의 임의의 이진 수열로 선형 이동한다. 선형이동은 아래와 같이 수행한다.

선형이동단계 : 임의 binary 수열 Q=(q_1, q_2, …, q_n) 선택

$0 < k < \min(a_i)$ for $q_i = 1$, 인 k 선택

public key 를 $e_i = a_i + kq_i$ 에 의해 계산

여기서 a_i는 위의 고밀도 알고리즘에 의한 hard knapsack

따라서 복호화키는 (A' , k, W, U)이 된다. 만일 수신자가 $S = \sum_{i=1}^{n} a_1 m_1$, 여기서 M=($m_1$, m_2, …, m_n)을 받으면 원래대로 기존의 방법에 의해 S를 복호화할 수 있다. 그러나 선형이동한 벡터 E로 메세지를 암호화한 정보를 다음과 같이 $S' = \sum_{i=1}^{n} e_1 m_1$를 받으면 다음의 식에 의거하여 복호화 해야 한다.

$$S \times W^{-1} = (\sum_{i=1}^{n} a_i m_i) \times W^{-1} \bmod U$$

$$= \sum_{i=1}^{n} (e_i + kq_i) m_i \times W^{-1} \bmod U$$

$$= S' \times W^{-1} + r \times \sum_{i=1}^{n} q_i m_i \bmod U$$

여기서 $r = kW^{-1} \bmod U$

위 식의 의미는 선형이동 벡터 E에 의해 암호화한 내용은 r 값과 임의 수열 Q 벡터와의 조합으로 이루어지므로 snap 연산이 적용되어 원래의 정보를 추출할 수 있는 경우가 한번 존재한다는 것이다. 고밀도 선형이동 knapsack을 통한 암호화 복호화 과정의 예는 다음과 같다.

암호 및 복호화 예 (key 생성 포함) : M=8657, w=311(w^{-1}=1608)인 경우

단계 1 : 임의의 초증가 수열 A' =(97, 123, 434, 697, 2491, 4453)

단계 2 : Hard knapsack 생성 A=(4196, 3625, 5119, 342, 4228, 8420)

단계 3 : 고밀도 knapsack 생성 H=(153, 204, 143, 31, 185, 23)

단계 4 : 복호화 키 생성 C=(84, 112, 418, 686, 2478, 4426)

여기서 Q=(1, 0, 1, 1, 1, 0)라 하고 k=29라 하면

단계 5 : 선형이동 벡터 E=(124, 204, 114, 2, 156, 23)를 얻는다.

여기서 메세지 벡터를 (1,0,1,0,1,0)라 하면 S' =394 이므로 $S_{tot} = S' * w^{-1} \bmod M = 2423$이고 $r = k*w^{-1} \bmod M = 5957$을 얻는다.

이것을 y=0,1,…, n까지 변화 시키며 추정해 보면 다음과 같이

$S_{tot} + 0 \times r = (2423 + 0 \times 5957) \bmod 8657 = 2423$

$S_{tot} + 1 \times r = (2423 + 1 \times 5957) \bmod 8657 = 8380$

$S_{tot} + 2 \times r = (2423 + 2 \times 5957) \bmod 8657 = 5680$

$S_{tot} + 3 \times r = (2423 + 3 \times 5957) \bmod 8657 = 2980$

를 얻게 되고 실제 해독되는 추정합은 y=3인 경우이다.

실제 기존의 Merkle Hellman 의 knapsack 은 이 알고리즘에서 모든 i에 대하여 $q_i = 0$인 특별한 경우이다. 본 알고리즘으로 생성된 key E가 knapsack 문제의 worst case 로 떨어질 확률은 1-(n/M/n!) 이상이다. 실

례로 n=100 이고 $M=2^{300}$ 일 때, 다른 초 증가 수열의 사상이 될 확률은 2^{-456} 보다 작고 임의의 수열의 사상일 확률은 10^{-95} 보다 작다.[5][6] 실제 이 선형이동 knapsack 의 공격 시험 결과 낮은 디멘젼에서도 선형이동 knapsack 에는 공격이 쉽게 적용되지 않고 디멘젼이 높아지면 해독될 확률이 급격히 적어지는 것으로 알려져 있다.[5][6]

Ⅲ. 선형이동 knapsack 을 위한 병렬구조

본 절에서는 제시된 선형이동 knapsack 암호화 시스템의 병렬구현을 위한 구조를 설명한다. 구조의 개념은 각 경로에서 circulation이 적어 전체 성능에 영향이 적은 것은 순차적으로 하고 복호화난 암호화와 같이 큰 영향을 미치는 부분은 동기식으로 병렬 수행할 수 있는 것이다. 전체 구조는 그림 1과 같다.[5]

전체 구성은 $S''=S+yr$를 계산하는 추정합계 연산블럭, 위의 S'' 를 받고 비밀키 A를 이용하여 snap의 연산을 수행하는 복호화 블럭, 공개키 E를 사용하여 누산을 통해 암호화하는 암호화 블럭, 원래 메세지인지 결정하는 결정블럭 그리고 각 블럭을 제어하기 위한 신호를 발생하는 제어 신호발생 블럭으로 구성한다.

Ⅳ. 시스템 구성 및 설계

본 장에서는 앞장에서 제시한 구조를 VLSI로 설계하는 내용을 설명한다. 제시한 것처럼 전체를 개념적으로는 추정 연산부, 복호화부, 암호화부, 결정부, 제어부로 구성되고 설계측면에서는 메모리 블럭, S′ 블럭, 복호화 블럭, 암호화 블럭, 결정 블럭, 제어 신호 발생 블럭으로 나누어 설계하였다.[5]

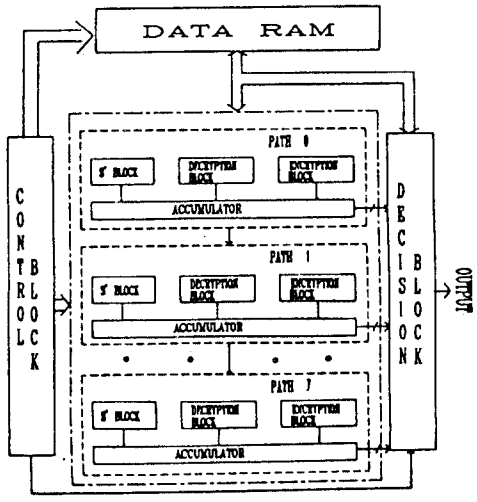


그림 1. 병렬 처리를 위한 전체 구성
Fig 1. Architecture for parallel processing

또 메모리 셀, 감지증폭기, 롬 및 가산기, 기타 기본 회로에 대하여 회로 시뮬레이션을 수행하였다.

레이아웃은 필요한 기본셀을 규준화하여 설계하고 이것을 바로 상위 레벨의 소단위 기능블럭에 대하여 사용하고 이렇게 설계된 기능 블럭들을 다음 상위 레벨에서 다시 기본 블럭으로 사용하여 그 상위레벨을 위한 설계에 이용하는 계층적인 방식으로 설계하였다.

1. 레지스터 블럭

암호화와 복호화 과정에서 필요한 초증가 knapsack 벡터와 선형이동 knapsack 벡터를 일시적으로 저장한 후 위하는 연산 수행시에 판독하여 계산을 수행하는 저장 레지스터로써 메세지 비트의 크기에 따라 레지스터의 크기도 달라진다. 메세지 비트가 8비트일 경우에는 두 knapsack 벡터가 각각 8 워드 × 16 비트의 크기를 차지하고 메세지 비트가 16 비트일 경우에는 두 knapsack 벡터는 16 워드 × 32 비트의 크기를 차지한다. 이러한 레지스터는 6개의 트랜지스터로 구성된 SRAM 셀을 사용하였고 행 디코더와 열 디코더, 감지증폭기 및 입·출력 구동회로로 구성된다. 본 설계에서는 8 워드 × 16 비트의 RAM을 2개 사용하여 하나는 복호화를 위한 비밀 키 초 증가 수열 A를 저장하게 하였고 다른 하나는 공개키 E를 저장하고 있다. r값은 레지스터에 직접 저장 하도록 하였다.

1.1 메모리 셀

데이타 RAM셀은 6개의 트랜지스터로 구성된 SRAM 셀을 사용하였다. 데이타 쓰기 동작은 칼럼 디코더가 선택되고 선행 충전이 LOW이면 한 비트선은 HIGH로 하고 다른 선은 LOW로 해서 데이타를 저장한다. 판독은 두 비트 선을 HIGH로 선행충전하고 위하는 워드선을 선택한다. 셀의 어느 한 비트는 LOW로 한다. 이때 두 비트선은 전압차를 감지 증폭기가 감지하여 셀의 데이타를 판독한다.

1.2. 감지 증폭기

감지 증폭기는 두 비트선의 작은 입력차를 감지하고 증폭시켜서 메모리의 억세스 시간을 줄인다. 먼저 메모리 셀의 데이타를 읽기 전에 두 비트선을 HIGH로 선

행 충전 시킨다. 다음에 선행 충전된 회로가 차단되고 셀이 선택되면 두 비트중 하나는 0이 되고 다른 하나는 1로 된다. 셀 내에 있는 트랜지스터는 매우 작고 BIT와 BIT′ 선의 커패시턴스는 크기 때문에 두 선중 한 선을 완전히 풀-다운 시키는 데는 많은 지연 시간이 필요하다. 따라서 감지 증폭기는 비트와 비트바선 사이의 작은 전압차를 감지하여 증폭시킴으로써 읽기 동작 시간을 감소시켜 억세스 시간을 줄인다.[11]

회로 동작은 먼저 비트선과 비트 선이 모두 똑같은 전압으로 선행 충전 되었을 때 왼쪽과 오른쪽 경로에는 같은 양의 전류 I가 흐른다. 예를 들어 비트선이 방전하기 시작하면 비트 쪽의 n 트랜지스터의 Vgs가 상대적으로 증가하여 2I가 흘러 출력은 LOW가 된다. RAM의 비트선이 약 2pF의 커패시턴스를 갖고 있다고 가정하고 LOAD에 0.2pF를 주고 간단한 RAM회로에 SENSE AMP를 달아 읽기 동작을 시뮬레이션 하였다. BIT 나 BIT′ 선이 선행충전되고 난 후 0을 읽을 경우 그림 2에 나타낸 시뮬레이션 결과에서 처럼 BIT 와 BIT′ 선의 전위차가 2V이면 충분히 감지함을 알 수 있고 본 회로 내부에서는 약 6ns의 억세스 시간을 갖는다. 그러나 메세지 비트의 증가에 따라 RAM 셀이 많아 지면 이 억세스 시간은 다소 증가할 것이다.

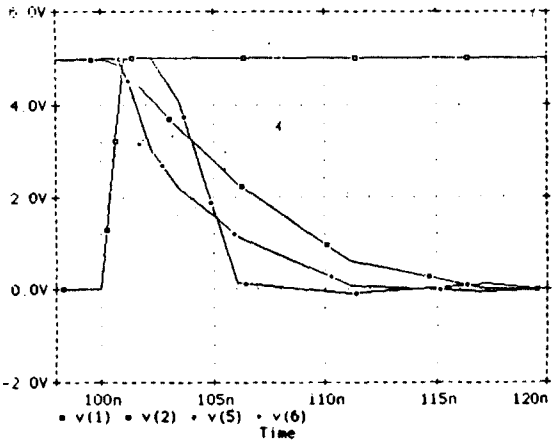


그림 2. 감지 증폭기를 갖는 RAM 회로 시뮬레이션
Fig 2. Simulation of RAM circuit with Sense Amplifier

※ 범례

V(1) : READ 신호 V(2) : BIT 선

V(5) : BUFFER 출력

V(6) : SENSE AMP 차동 출력단

1.3 어드레스 디코더

어드레스 디코더는 읽기나 저장할 때 필요한 회로로써 필요한 데이타의 어드레스를 입력으로 받아서 거기에 해당하는 워드선을 HIGH로 만들어 준다. 복호화와 암호화하는 과정에서 사용된 어드레스 디코더는 4개의 어드레스를 입력으로 받아서 24개의 워드선 중에서 1개의 워드선을 선택한다. 선행 충전된 비트선을 고려할 때 단순한 NOR 게이트를 사용하지만 워드선을 구동하기 위해서는 버퍼링이 필요하게 되는데 이런 상황에서 NOR구조는 한개를 제외하고는 모든 디코더를 액티브하게 한다. NOR게이트를 사용할 경우 CMOS기술에서 부가적인 문제점은 직렬 P 트랜지스터 구조로 인하여 워드선의 상승시간이 증가하고 디코더의 면적을 넓히게 된다. 따라서 이런 문제를 피하기 위해 NAND 게이트를 사용하고 워드선 드라이버로써 NOR 게이트를 사용한다.

2. 제어 블럭

시스템 제어부는 전체 knapsack 시스템의 각 블럭에서 필요한 제어신호를 발생시켜 적절한 순간에 공급해 주는 역할을 수행한다. 일반적으로 제어부를 구현하는 방법으로써 랜덤 논리 구현과 구조화된 논리 방법이 있다. 랜덤 논리 구현은 카운터와 간단한 게이트(NAND, NOR, INVERTER)와 저장 요소인 플립 플롭을 이용한 순서 논리 회로를 사용하지만 원하는 제어 신호의 타이밍 도를 얻기 위해서는 설계가 복잡한 면이 있다. 하지만 랜덤 논리 구현은 구조화된 논리 구현 방법이 불필요한 회로도 포함하는 반면에 필요한 논리 회로 만으로 구성되므로 칩 면적이 감소된다. 이러한 랜덤 논리 구현은 논리, 회로 설계 및 레이 아웃 등을 수행하는데 많은 시간이 요구되고 오류를 정정하는데 많은 노력이 요구되므로 설계시간이 짧고 오류를 정정하는데 장점을 지닌 규칙적이고 설계가 용이한 구조화된 논리인 ROM으로 제어부를 설계하였다.

ROM제어부의 구성은 다음과 같다. 외부 핀으로 부터 시스템의 초기화 신호가 입력되면 시스템의 초기화 후 순차적인 어드레스를 발생시키는 프로 그래밍 카운터, 4 입력 NAND 게이트도 설계된 어드레스 디코더, 전체 시스템에 필요한 제어 신호를 발생 시키는 ROM 배열로 구성된다. ROM은 24 워드 × 34 비트로 구성되어 있다. 전체 블럭도를 그림 3에 보였다. ROM의 각 워드는 주기적으로 이루어지는 제어신호의 한 스텝에 해당되고 각 비트는 제어 신호의 종류를 의미한다. 제

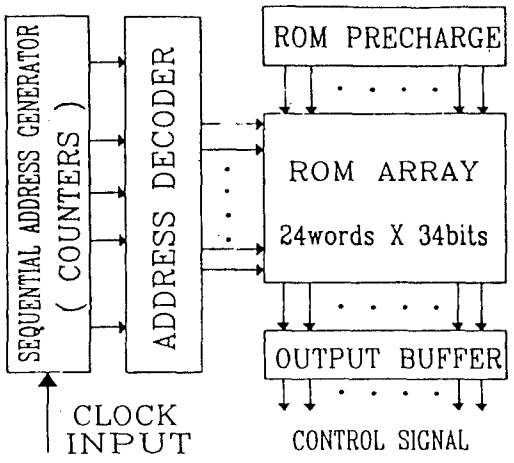


그림 3. 제어부 블럭도.
Fig 3. Block diagram of control part

표 1. 제어부 ROM 내용
Table 1. The content of control part

어드레스	데 이 타 M[0:34]
00000	10000 00000 00000 00000 00000 00000 0000
00001	01000 00000 00000 00000 00000 00000 0000
00010	00100 00000 00000 00000 00000 00000 0000
00011	00010 10000 00000 00000 00000 00000 0000
00100	00010 01100 01100 00000 00000 00000 0000
00101	00010 01010 01111 00000 00000 00000 0000
00110	00010 01001 01111 10000 00000 00000 0000
00111	00001 00000 11111 11110 00000 00000 0000
01000	00001 00000 01111 11111 00000 00000 0000
01001	00001 00000 01111 11111 00000 00000 0000
01010	00001 00000 01111 11111 00000 00000 0000
01011	00000 00000 01111 11111 00000 00000 0001
01100	00000 00000 00111 11111 11000 00000 0000
01101	00000 00000 00101 11111 10011 00000 0000
01110	00000 00000 00101 01111 10010 01100 0000
01111	00000 00101 01001 10010 10010 01001 1000
10000	00000 00101 01010 10010 10010 01001 0010
10001	00000 00101 01010 10010 10010 01001 0010
10010	00000 00101 01010 10010 10010 01001 0010
10011	00000 00101 01010 10010 10010 01001 0010
10100	00000 00000 00001 01010 00110 01001 0010
10101	00000 00000 00000 01010 00000 01001 0010
10110	00000 00000 00000 00010 00000 00011 0010
10111	00000 00000 00000 00000 00000 00000 0110

어신호의 성격으로는 누산기의 동작 모드를 결정하기 위한 신호, 추정 연산기의 결과에 대한 경로를 열기 위한 신호, 데이터 RAM으로 부터 신호를 읽기 위한 신호, 각 블럭 간 데이타를 이동 시키기 위한 신호, 암호화 및 복호화 블럭의 누산 동작을 제어하는 신호로 구별된다. ROM내용은 표 1과 같고 각 제어 신호의 기능은 표 2와 같다.

표 2. 제어신호의 기능
Table 2. The function of control signal

제어신호	제 어 신 호 내 용
S	외부에서 S계수를 받아서 레지스터에 저장
R	외부에서 r계수를 받아서 레지스터에 저장
S'	메세지메터 연산함S를 받아서 레지스터에 저장
A	승중치 수열A를 받아서 RAM에 저장
E	선형이동 메터 E를 받아서 RAM에 저장
S_DATA	추정 연산함S를 계산하기 위해 S계수를 덧셈기에 입력시킴
R_DATA	r계수값들을 연속해서 덧셈기에 출력시킴
S1	덧셈기를 동작시켜 PATH 1으로 결과를 출력시킴
S2	덧셈기를 동작시켜 PATH 2으로결과를 출력시킴
S3	덧셈기를 동작시켜 PATH 3으로결과를 출력시킴
S4	덧셈기를 동작시켜 PATH 4으로결과를 출력시킴
B1	메모리로부터 계수를 읽어들여 감산기에 입력시킴
SH1	메세지 레지스터를 오른쪽으로 shift시킴
A2	A1레지스터로부터 계수를 읽어들여 감산기에 입력시킴
SH2	PATH 2의 메세지 레지스터의 값을 오른쪽으로 SHIFT시킴
A3	A2레지스터로부터 계수를 읽어들여 감산기에 입력시킴
SH3	PATH 3의 메세지 레지스터의값을 오른쪽으로 shift 시킴
A4	A3레지스터로부터 계수를 읽어들여 감산기에 입력시킴
SH4	PATH 4의 메세지 레지스터의값을 오른쪽으로 shift 시킴
FB1	덧셈기의 출력을 입력으로 궤환시킴
E1	메모리로부터 계수를 읽어들여 가산기에 입력시킴
E1_ACK	덧셈기를 동작시켜 E1계수를 누적해서 S를 계산
S1_OUT	연산함이 입력된 S와 일치할 경우 그 메시지를 출력시킴
E2	E1레지스터로부터 계수를 읽어들여 가산기에 입력시킴
E2_ACK	덧셈기를 동작시켜 E2계수를 누적해서 S를 계산
S2_OUT	연산함이 입력된 S와 일치할 경우 그 메시지를 출력시킴
E3	E2레지스터로부터 계수를 읽어들여 가산기에 입력시킴
E3_ACK	덧셈기를 동작시켜 E3계수를 누적해서 S를 계산
S3_OUT	연산함이 입력된 S와 일치할 경우 그 메시지를 출력시킴
E4	E3레지스터로부터 계수를 읽어들여 가산기에 입력시킴
E4_ACK	덧셈기를 동작시켜 E3계수를 누적해서 S를 계산
S4_OUT	연산함이 입력된 S와 일치할 경우 그 메시지를 출력시킴
E4_BAK	덧셈기의 출력을 입력으로 궤환시킴
E_START	메모리로부터 E값을 읽어서 레지스터에 저장함

3. 추정 합계 연산 (S'')블럭

S'' 블럭은 추정 합계 연산 $S'' = S+jr$, 여기서 j= 0,1, … , y (본 연구에서는 y=4), 을 계산하는 블럭으로 S'' 를 순차적으로 누산해서 구한후 이 값들을 각각 다른 경로로 전송한다. 전체 블럭을 그림 4(a)에 보였다. 그림 4(b)는 계수 레지스터로 부터 R 레지스터 값과 S 레지스터 값을 받아서 가산과 누적을 수행하는 회로이다. 그림 4(c)의 신호도를 참조하여 볼 때 만약 S_ADD 신호가 1이면 S 레지스터 값이 전 가산기의 입력으로 연결되어 R 레지스터 값과 덧셈을 수행한다.

또 S_ADD 신호가 0이고 S_BACK 신호가 1이면 전 가산기의 이전 출력인 sum[n-1] 값이 피드백 되어 전 가산기의 입력으로 연결되므로 r과 sum[n-1]의 덧셈이 수행된다. 즉 S_ADD 신호가 1일 때는 각 입력값을 더하는 연산을 수행하고 S_ADD가 0일 때에는 누산하는 기능을 수행한다. 따라서 최초에 S 레지스터 값과 S_ADD 신호에 따라 초기 S″ 값을 구하고 그 이후로 R 레지스터 값과 S_BACK 신호를 받아 궤환하는 식으로 R 레지스터 값을 누적하여 더해 나간다.

매 클럭 마다 추정 연산 합 S_1'', S_2'', S_3'' …, S_n'' 이 차례로 누산되어 계산되며 이 값은 각각의 클럭 마다 y개의 경로로 전달되는데 이 과정은 그림 4(b)의 회로로 구현된다. 즉 각 클럭 마다 y개의 tri_state 회로를 사용하여 누산된 연산합을 차례로 이동시키고 제어 회로에 의한 신호로 래치에 저장한다. 래치에 저장된 연산합 S_n'' 는 시스템 신호에 따라서 출력 버퍼를 ON 시켜서 y경로의 데이타를 출력시킨다.

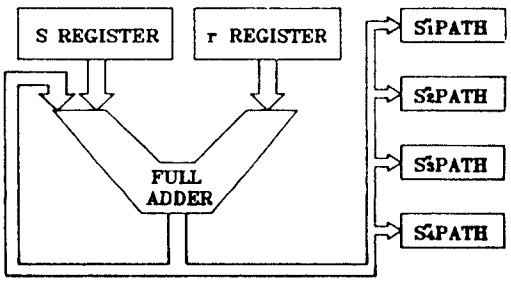


그림 4(a). S″ 블럭도
Fig 4(a). S″ block diagram

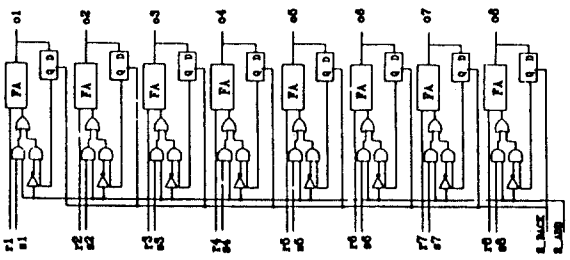


그림 4(b). S″ 블럭의 누적 가산 회로 구성도
Fig 4(b). Accumulator and adder circuit of S″ block

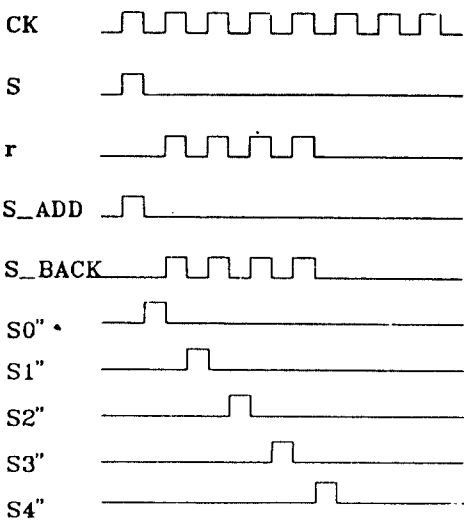


그림 4(c). S″ 블럭의 신호 타이밍 도
Fig 4(c). Signal timing diagram for S″ block
그림 4. S″ 블럭, 주요회로, 신호도
Fig 4. S″ block, main circuit, timing diagram

4. 복호화 블럭

복호화 블럭은 앞단의 S″ 블럭에서 구한 S″ 값과 RAM에 저장되어 있는 초 증가 knapsack 벡터 A를 순차적으로 읽어 들이고 조건 감산을 수행하여 메세지 벡터값 X를 구하는 블럭이다. 블럭도를 그림 5(a) 에 보였다. 이 메세지 벡터를 구하는 과정은 snap 연산으로 n번의 감산이 필요하다. Modulo-n의 하향 계수기를 이용하여 순차적으로 어드레스를 발생시키고 이 어드레스를 디코더하여 각각의 연산에서 필요한 초증가 벡터값을 A 레지스터에 저장한다. 이 저장된 값과 S″ 블럭에서 구한 S″ 값을 받아들여 먼저 비교 연산을 수행한다. 두 레지스터로 부터 읽어 들인값 S″ 와 A값에서 S″ 가 A보다 작다면 메세지 레지스터에 0을 출력 시키고 감산 연산을 수행하지 않고 이전 값으로 다음 연산을 수행한다. 여기에서 비교 연산을 수행하는 회로는 그림 5(b)와 같다. 비교하려고 하는 두 값 S″ 와 A값의 LSB의 합 연산에서 C_{-1}값을 1로 놓고 연산을 수행하여 마지막 MSB에서 캐리가 발생하면 S″ 가 A보다 크다는 것을 나타내고 같을 경우에는 각 비트의 합을 EXCLUSIVE-NOR 한 값이 1일때이다. 즉 이때는 메세지 레지스터에 1 값을 출력함과 동시에 궤환 클럭(feedback clock)을 동작시켜 각 비트 값을 궤환시킨다. 그렇지 않고 캐리가 발생하지 않고 합의 EXCLUSIVE

-NOR 한 값이 0 일 경우에는 메세지 레지스터에 0 값을 출력시키고 궤환 클럭을 동작시키지 않음으로써 그 값이 다음 연산에 그대로 사용되게 된다.

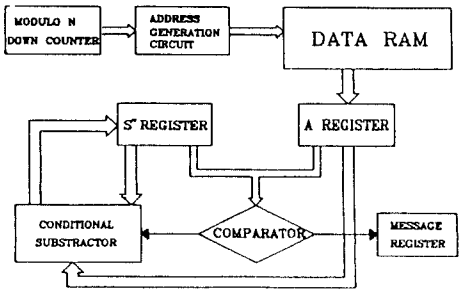


그림 5(a). 복호화 블럭도
Fig 5(a). Decryption block diagram

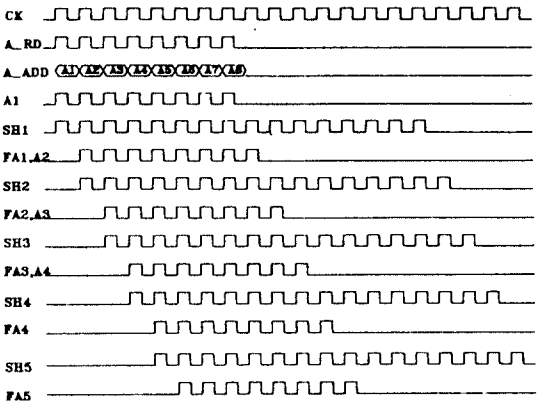


그림 5(c). 복호화 블럭의 신호 타이밍 도
Fig 5(c). Signal timing diagram for decryption block

그림 5. 복호화 블럭, 부요회로, 신호도
Fig. 5. Decryption block, main circuit, timing diagram

5. 암호화 블럭

암호화 블럭은 전단의 복호화 블럭에서 구한 메세지에 대해서 선형 이동 knapsack 벡터를 이용하여 다시 암호화 하는 블럭이다. 그림 6(a)에 암호화를 위한 블럭도를 보였다. 이는 각각의 경로에 대해 암호화를 수

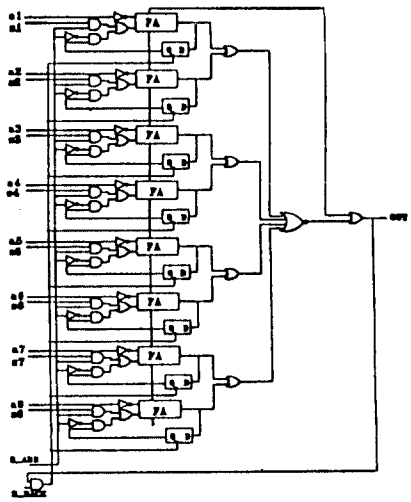


그림 5(b). 복호화 블럭의 조건 감산 및 비교 회로
Fig 5(b). Conditional substraction and comparison circuit of decryption block

행함으로써 각 경로의 암호화된 S′ 값과 원래의 수신된 S′ 값을 비교하기 위함이다. 이 비교를 통해 어느 경로에서 올바로 복호화를 수행했는지를 결정할 수가 있다. 이는 아래의 회로를 이용하여 수행하는데 전단의 복호화 과정에서 구한 각 메세지 비트를 직렬로 출력시키고 RAM에 저장되어 있는 선형 이동 knapsack 벡터 E를 순차적으로 읽어 들임으로써 S′ =Σex를 계산하는 블럭이다.

복호화 블럭과 마찬가지로 암호화 블럭에서도 n번의 합산이 필요한데 이를 위해 modulo n의 하향 계수기를 사용하여 순차적으로 어드레스를 발생시키고 이 주소를 디코딩하여 각각의 연산에서 필요한 선형 이동 knapsack 벡터 값을 E 레지스터에 저장한다. 또 전단의 복호화 블럭에서 구한 메세지 벡터가 SIPO(Serial Input Parallel Output)레지스터에 저장되어 있는데 이를 한 비트씩 쉬프트 시킴으로써 출력된 메세지 비트가 1일 경우와 0일 경우 동작을 달리 하도록 한다. 즉 메세지 레지스터에서 직렬로 출력된 값과 제어 신호를 AND 게이트를 이용하여 메세지 비트가 1 일 경우에는 E 레지스터의 값을 가산기에 입력시켜 덧셈 연산을 수행하고 메세지 비트가 0 일 경우에는 덧셈 연산을 수행하지 않고 그 값을 그대로 저장한다. 차례로 누적시켜 가며 행하는 덧셈 연산은 S″ 블럭에서 행한 것과 비슷하다. 즉 이 블럭에서는 선형 이동 knapsack 벡터의 값과 누적되어 피드백 되는 S″ 값을 더함으로써 최종적인 S′ 값을 S′ 레지스터에 저장시킨다. 이 S′ 값은 다음 결정 블럭에서 원래 수신된 S′ 값과 비교를 행하게 된다.

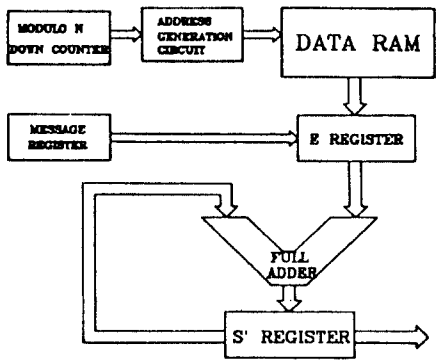


그림 6(a). 암호화 블럭도
Fig 6(a). Encryption block diagram

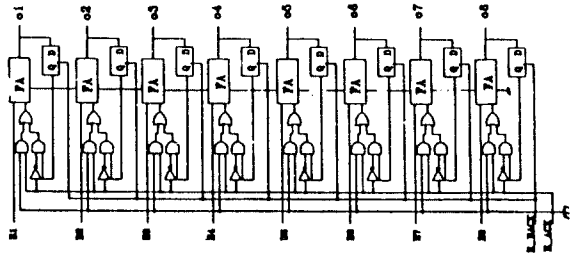


그림 6(b). 암호화 블럭의 누산기 회로 구성도
Fig 6(b). Accumulator circuit of encryption block

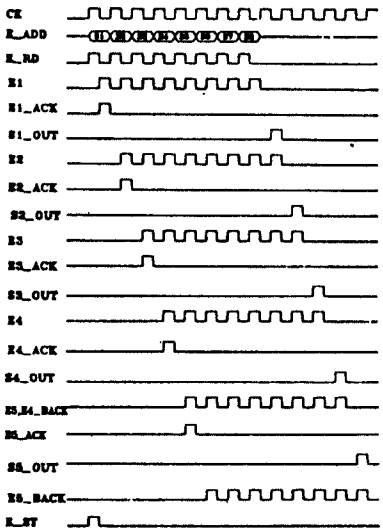


그림 6(c). 암호화 블럭의 신호 타이밍 도
Fig 6(c). Signal timing diagram for encryption block

그림 6. S″ 블럭, 주요회로, 신호도
Fig 6. S″ block, main circuit, timing diagram

6. 결정 블럭

결정 블럭은 여러 경로에 대해 수행한 암호화 값이 원래 수신된 메세지 합 S′ 와 비교를 행함으로써 어느 경로에서 올바로 암호화와 복호화가 수행됐는지를 결정하여 일치하는 경로의 메세지 값을 출력시키는 회로이다. 그림 7에 블럭도를 보였다. 구성은 여러 경로 중 올바른 메세지 데이타를 출력시키는 것을 선택하는 MUX 회로와 비교기 회로 구성된다. 비교 기능은 단순한 XOR 연산을 수행함으로써 구현된다. 각 경로에 대해 메세지 레지스터에 저장된 값을 출력시키는 제어 신호를 비교기 출력값과 AND 연산을 행함으로써 비교기 출력이 1이면 그 경로에 메세지를 출력 시키고 비교기의 출력값이 0 이면 그 경로의 메세지 레지스터는 동작을 하지 않는다.

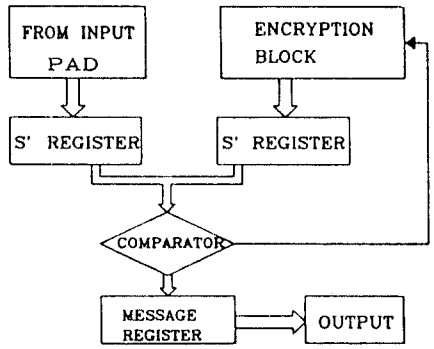


그림 7(a). 결정 블럭도
Fig 7(a). Decision block diagram

V. 시스템 성능

칩으로 구현한 프로세서 시스템은 그림 8과 같다. 전체 칩 면적은 $2550\times2560\mu m^2$ 을 차지하였다. 구현한 시스템의 성능을 정량적으로 측정해보기 위해 각 블럭간 연산 사이클과 지연시간을 구해보면, 첫째로 추정합계 연산 블럭에서는 기본 추정 값과 y개 만큼의 추정치를 계산하기 위한 연산이 존재한다. 각 추정치 계산은 플립플롭, AND 게이트, OR 게이트, 인버터, 전가산기의 지연을 합한 것만큼 지연이 있게 된다. 이것은 임의의 n비트에서 $(0.5n+1)\ \tau_s$ 로 일반화 할 수 있다.

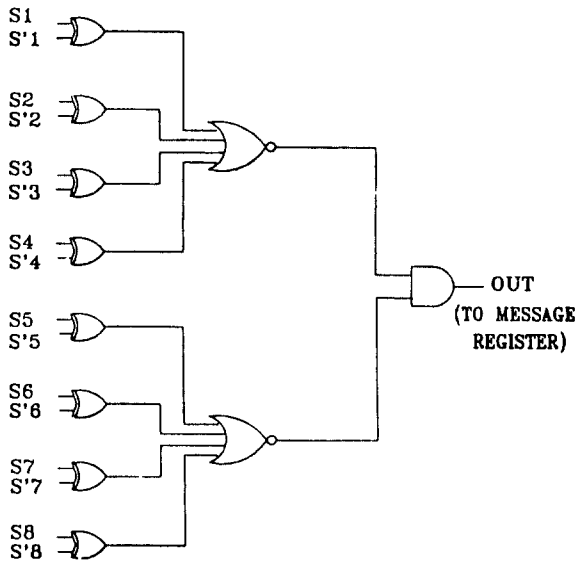


그림 7(b). 결정 블럭의 비교 회로
Fig 7(b). Comparison circuit of decision block

그림 7. 결정 블럭 및 주요회로
Fig 7. Decision block and main circuit

여기서 τ_{S} 는 위에서 설명한 추정치 계산 지연이다. 둘째로 복호화 블럭의 지연인데, 이것은 RAM 으로 부터 n개 데이타의 판독 시간(복호화를 위한 연산도 포함됨)과 계산된 비트의 암호화 블럭으로 쉬프트 시간을 합한 것인데 이것은 암호화 블럭에서 중복되므로 생략하여 $n\tau_{RAM}$ 로 나타낼 수 있다. 여기서 τ_{RAM} 은 RAM 억세스 시간이다. 셋째로 암호화 블럭으로 역시 RAM 억세스 시간, 플립플롭에서의 시간으로 전체 $n\tau_{RAM}$ + $(0.5n)\ \tau_{ff}$ 의 시간이 소요된다. 여기서 τ_{ff} 는 플립플롭의 지연시간이다. 넷째로 결정 블럭에서 결정을 위한 시간은 플립플롭, XOR 게이트, NOR 게이트, AND 게이트 에서의 지연시간과 최종 메세지 레지스터가 데이타를 내놓는 시간으로 $(0.5n+1)\tau_{clock}+\tau_{ff}$의 시간이 소요된다.

각 기본 셀들의 지연시간을 시뮬레이션으로 구해보면 기본 게이트들은 몇 개를 합쳤을 때 1 ns, 가산기와 플립플롭은 약 2ns, RAM의 판독 시간은 약 6ns의 시간이 소요되므로 이로 전체 지연을 계산할 수 있다. 본 설계에서는 RAM 판독시간을 기본 클럭의 주기로 생각하여 구성하였으므로 이를 τ라 하고 추정 연산합 이전의 W^{-1} 곱 연산 시간을 제외 한다면 전체 지연은 $(3n+2)\tau$가 된다. 따라서 τ를 6ns로 잡으면 n=8 인 경우 초당 6.4×10^{6} 의 정보를, n=100 이면 약 550K (100 비트 단위)의 정보를 처리할 수 있다. 이것은 다른 성질의 공개키 암호화 시스템과 비교하면 RSA 프로세서는 512비트 혹은 1024 비트에 대해 5kb/s-64kb/s 의 처리속도를[7][12], 유한체 ($GF(2^{155})$에서 동작하는 DEP 공개키 프로세서[10]는 500kb/s의 속도를 갖고 동작한다. 본 시스템에서 W^{-1}곱 연산은 단순한 곱셈만이 필요하므로 모듈로 연산을 수행하여 크게 시간을 요하지 않으므로 이 연산 시간을 제외한다면 n=200 의 경우 30-60배 이상 속도가 빠름을 알 수 있다. 기존의 Merkle Hellman의 knapsack 시스템을 구성할 경우 암호화를 위해서 n번 이하의 데이타 전송과 덧셈이 필요하고 복호화를 위해서는 n번 이하의 비교 및 감산만으로 이루어지므로 시간, 면적에서 매우 우수하나 실제 시스템 자체가 해독이 용이하다. 선형이동 시스템은 각 추정합에 따라 경로에서 복호 및 암호화를 수행하여 원래의 메세지인지를 확인하므로 기존의 M-H 시스템이 n/2 개가 구현되어야 하는 것이 특징이다. 빠른 속도를 위해서는 n/2개의 복호 및 암호회로를 모두 구현해야 하고 실리콘 면적을 줄이기 위해서는 복호 및 암호회로를 공유하여 속도를 희생해야 한다. 따라서 최적 속도를 갖는 bit 확장이 가능하도록 병렬성을 위한 효과적인 구조가 필요하다.

VI. 결론

구현된 공개키 시스템은 선형 이동 knapsack 암호화 시스템이다. 이의 알고리즘의 설명을 위해 Merkle Hellman 의 knapsack 이나 기타 여러 knapsack 의 약점을 제시하고 생성된 암호키가 1대 1임을 보장 못하는 것이 약점이지만 위의 기존 knapsack 에 대한 단점을 보완할 수 있는 고밀도 knapsack 과 선형이동 knapsack 시스템을 설명하고 이 선형 이동 knapsack 시스템의 구현을 위해 병렬구조를 제안하였다. 본 프로세서 시스템의 중요 내용은 면적 속도를 최대한 고려하여 많은 반복으로 다량의 시간이 요구되는 부분에 대해 파이프 라인식으로 병렬처리 하도록 구성하였다. 이것은 지수형 공개키 시스템(exponentiation cipher)보다 30-60배의 더 빠른 속도가 나올 것으로 사료된다.

각 기능 블럭들을 계층적으로 설계 하였고 각 블럭들에 대해 논리 시뮬레이션 및 필요 회로에 대해서는 회로 시뮬레이션을 수행하여 기능 검증 및 성능 평가

에 사용하였다.

차후 산업용 레이아웃 생성기를 위한 프레임 웍에의 연결, 완전한 설계규칙 검사, 자동 셀 생성기의 지원에 대한 연구와 본 암호화 시스템의 비트 확장을 위한 구조 및 자동 생성에 관한 연구가 기대된다.

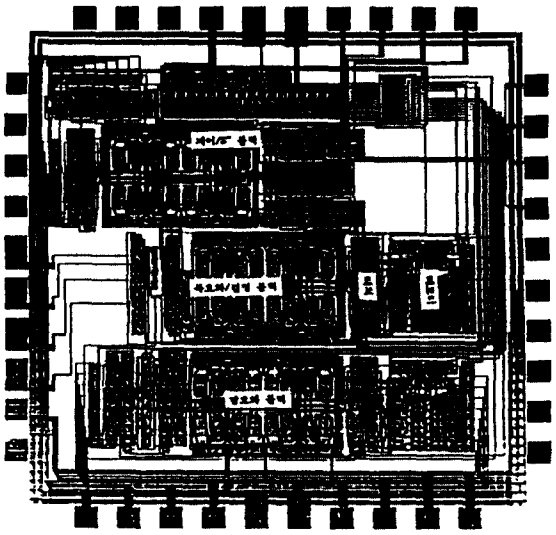
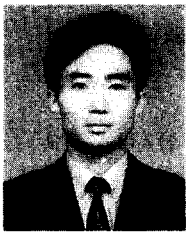


그림 8. 설계된 암호화 칩
Fig 8. Cryptography chip desigmed

참 고 문 헌

1. Dorothy Elizabeth Robling Denning, "Cryptography and Data Security", Addison-Wesley Publishing Co., 1982.
2. Y.G. Desmedt, J.P. Vandewalls, etc, "A Critical Analysis of the Security of Knapsack Public-Key Algorithms", IEEE Trans. on Infor. Theory, Vol. IT-30, No. 4, pp 601-611, July 1984.
3. 김세헌, 엄봉식, "Knapsack 공개키 암호법의 비교 분석 및 개선방법 개발", 1991 년도 데이타 보호 기술 WORKSHOP 논문집, 1991.
4. E.F. Brickell and A.M. Odlyzko, "Cryptanalysis: A Survey of Recent Results", Proc. of IEEE, Vol. 76, No. 5, pp 578-593, May 1988.
5. 차균현, 백인천, "선형 이동 Knapsack 공개키 암호화 시스템의 구현에 관한 연구", 한국통신학회 논문지 제 16권 제9호, pp 883-892, 1991.
6. Chi-Sung Laih, Jau-Yien Lee, etc, "Linearly Shift Knapsack Public-Key Cryptosystem", IEEE Journal of selected area in Comm. 1989.
7. Andre Vandemeulebroecke, etc, "A single chip 1024 bits RSA processor", Eurocrypt, 1989.
8. W. Geiselmann, D. Gollmann, "VLSI Design for Exponentiation in GF(2^n)", AUSCRYPT 90, pp 321, 1990.
9. Tony Rosati, "A High Speed Data Encryption Processor for Public Key Cryptography", IEEE CICC, pp 12.3.1-12.3.5, 1989.
10. P. Andrew Scott, "Architectures for Exponentiation in GF(2^m)", IEEE Jounal on selected areas in comm., 1988.
11. N. Weste, Kamran Eshranghian, "Principles of CMOS VLSI Design", Addison Wesley, 1985.
12. Peter A Ivey, etc, "An Ultra-High Speed Public Key Encryption Processor", IEEE CICC, pp 19.6.1 19.6.4, 1992.



白 寅 天 (In Cheon Paik) 정회원

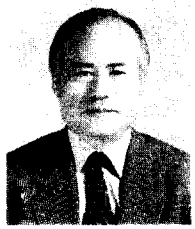
1985년 2월 : 고려대학교 전자공학과 공학사

1987년 2월 : 고려대학교 전자공학과 공학석사

1992년 8월 : 고려대학교 공학사 공학박사

1993년 ~ 현재 : 순천향대학교 전산통계학과 전임강사

※주요관심분야 : VLSI설계를 위한 CAD, 객체지향 소프트웨어 설계



車均鉉 (Kyun Hyon Tchah) 정회원

1965년 : 서울 대학교 공학사

1967년 : 미국 일리노이 대학교 공학석사

1976년 : 서울 대학교 공학박사

1977년 ~ 현재 : 고려 대학교 전자공학과 교수