IMT-2000 시스템을 위한 기밀성 및 무결성 하드웨어 구현

정회원 최 용 제*, 김 호 원**, 김 무 섭***, 정 교 일****

Hardware Implementation of Confidentiality and Integrity Algorithms for IMT-2000 System

Yong-je Choi*, Ho-won Kim**, Moo-seop Kim***, Kyo-il Chung**** *Regular Members*

요 약

본 논문에서는 IMT-2000 시스템을 위한 기밀성 및 무결성 하드웨어를 구현하였다. 구현된 하드웨어는 기밀성 알고리즘과 무결성 알고리즘을 모두 수행할 수 있도록 설계되었으며, 파이프라인 구조의 KASUMI 하드웨어를 사용하여 높은 주파수에서 동작할 수 있도록 하였다. 또한 외부와의 통신을 위한 PCI 인터페이스 모듈을 내장하여 실제 시스템으로 응용이 용이하도록 하였다. 설계된 하드웨어는 FPGA 보드로 구현되어 테스트되었으며, 테스트 결과 33MHz 동작주파수에서 최대 330Mbps의 암·복호율의 성능을 얻을 수 있었다.

ABSTRACT

In this paper, we designed the hardware of confidentiality and integrity algorithms for IMT-2000 system. The hardware is implemented to be able to calculate both confidentiality and integrity algorithms, and a pipelined KASUMI hardware is used for a core operator to achieve high operation frequency. And it has a PCI interface logic to communicate with a host system, and which makes our hardware to be easily applicable to practical environments. We tested our hardware on the FPGA board which we have designed, and its peak performace is about 330 Mbps encryption or decryption rate under 33 MHz clock frequency.

Ⅰ. 서 론

진화된 GSM(Global System for Mobile communication) 핵심망과 UTRA(Universal Terrestrial Radio Access) 무선 접속기술을 바탕으로 한 3세대 이동통신망인 IMT-2000 시스템은 보안성을 강화하기 위해서, 인증을 위한 난수 발생 알고리즘인 f0를 비롯하여 가입자의 네트워크 인증을 위한 f1 알고리즘, 재동기 메시지 인증을 위한 f1* 알고리즘, 가입자 인증을 위한 f2 알고리즘, 무선 구간 암호화 키 생성을 위한 f3 알고리즘, 무선 구간 무결성 키 생성을 위한 f4 알고리즘, 가입자 익명성 키 생성을 위한 f5 알고리즘, 재동기 익명성 키 생성을 위한 f5* 알고리즘 등을 정의하고 있다[3].

또한, 단말기와 RNC(Radio Network Controller) 시스템 사이의 보안을 위하여 기밀성 알고리즘 f8과 무결성 알고리즘 f9을 정의하고 있다[1]. 두 알고리즘은 모두 KASUMI[2] 암호 알고리즘에 안전성의 근간을 두고 있다. KASUMI 암호 알고리즘은 일본 미쓰비시의 MISTY[5] 암호 알고리즘을 일부 수정한 블록 암호 알고리즘으로, 통신 시스템의 성능 향상을 위해서는 하드웨어 구현이 필요하다. 특히 RNC 시스템의 경우 수십에서 수백 채널의 데이터에 대한 처리가 이루어져야 하기 때문에, 암호 알고리즘

* 한국전자통신연구원 정보보호기반연구팀(choiyj@etri.re.kr)
** 한국전자통신연구원 정보보호기반연구팀(khw@etri.re.kr)
*** 한국전자통신연구원 정보보호기반연구팀(gomskim@etri.re.kr)
**** 한국전자통신연구원 정보보호기반연구부(kyoil@etri.re.kr)
※ 본 논문은 2002년 4월 JCCI 학술대회에서 우수논문으로 선정되어 게재 추천된 논문입니다.

을 고속으로 처리할 수 있는 하드웨어 구현이 필수적이다.

본 논문은 이러한 IMT-2000 시스템의 기밀성 및 무결성 알고리즘을 위한 하드웨어 구현에 관한 것으로, 다음과 같은 구성으로 되어 있다. 2절에서는 두 알고리즘의 핵심인 KASUMI 암호 알고리즘 특성 및 하드웨어 구현에 대하여 언급한다. 3절에서는 기밀성 및 무결성 알고리즘 특성 및 하드웨어 및 시연 시스템 구현에 대하여 언급하고, 4절에서 결론을 맺는다.

Ⅱ. KASUMI 암호 알고리즘

본 절에서는 KASUMI 암호 알고리즘을 살펴보고, 여러 가지 구현 기법에 대하여 언급하였다.

1. KASUMI 암호 알고리즘

KASUMI 암호 알고리즘은 8 라운드 Feistel 구조를 갖는다. 기본 구조는 [그림 1]과 같다[2].

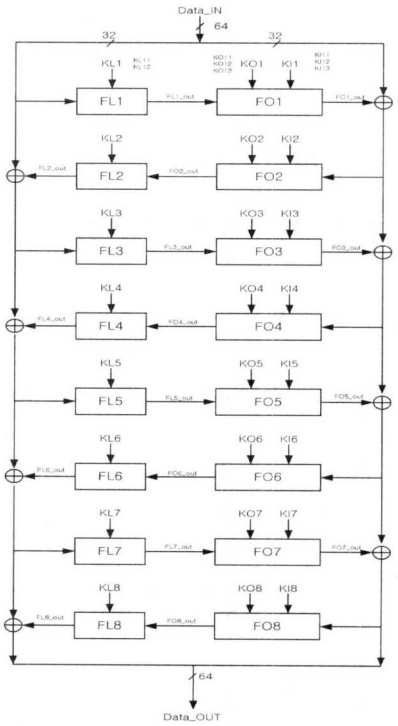


그림 1. KASUMI 암호 알고리즘 기본 구조

이는 64비트 입력 데이터 블록에 대하여 동작하고 128 비트 키 값을 갖는다. KASUMI 알고리즘에 들어가는 64비트 입력 데이터는 32비트 크기의 L_0와 R_0로 나뉘어진다. 라운드를 i라고 했을 때, KASUMI 알고리즘의 동작은 다음 식과 같다.

$$R_i = L_{i-1},\ L_i = R_{i-1} \oplus f_i(L_{i-1}, RK_i) \quad (1)$$

여기서, fi는 라운드 함수이며, RKi 는 i 번째 라운드의 라운드 키 값이다. KASUMI 알고리즘의 최종 출력은 64비트 길이의 $L_8 \| R_8$으로 만들어 진다. KASUMI 알고리즘의 각 라운드를 구성하는 fi 함수는 FL 블록과 FO 블록으로 구성되며, 32비트 입력에 대한 32비트 출력을 갖는다. 또한 라운드 키 값 RK_i를 필요로 하는데, RK_i 라운드 키는 (KL_i, KO_i, KI_i)로 구성된다. f_i 함수는 짝수번째 라운드이냐, 아니면 홀수번째 라운드이냐에 따라 다른 구조를 갖는다. 1,3,5,7 번째 라운드에서의 f_i 함수는 다음과 같다.

$$f_i(I, RK_i) = FO(FL(I, KL_i), KO_i, KI_i) \quad (2)$$

2,4,6,8 번째 라운드에서의 f_i 함수는 다음과 같다.

$$f_i(I, K_i) = FL(FO(I, KO_i, KI_i), KL_i) \quad (3)$$

FL 블록은 32비트 데이터와 두개의 16비트 서브키에 대하여 32비트 출력을 갖는다. FL 블록의 구조는 [그림 2]와 같다.

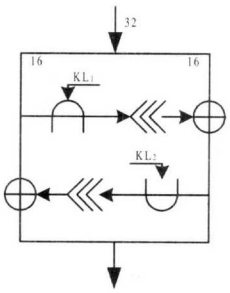


그림 2. FL 블록 구조

FL 블록은 다음과 같은 식으로 표현할 수 있다. 여기서, R과 L은 16비트 길이의 입력값을 의미하고 ROL은 1bit left rotation을 의미한다. 출력은 32비트 $L' \| R'$ 값이다.

$$R' = R \oplus ROL(L \cap KL_1) \quad (4)$$

$$L' = L \oplus ROL(R' \cup KL_2) \quad (5)$$

FO 블록의 구조는 [그림 3]과 같다.

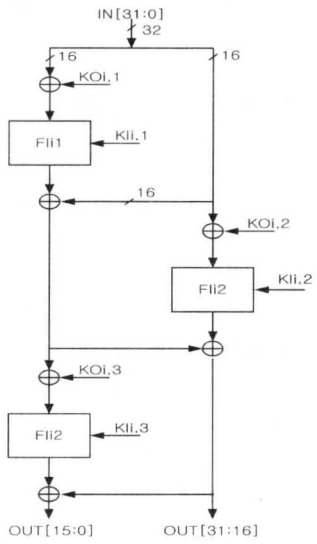


그림 3. FO 블록 구조

그림에서 알 수 있듯이 FO 블록은 32비트 길이의 입력 데이터와 48비트 길이의 KO_i, KI_i 서브키로 구성된다. 32비트 입력값 i는 16비트 길이의 L_0와 R_0로 나눠진다. 그리고 48비트 길이의 서브키는 16비트 길이의 키로 다시 나눠진다. 이를 식으로 표현하면 다음과 같다.

$$KO_i = KO_{i,1} \| KO_{i,2} \| KO_{i,3} \quad (6)$$

$$KI_i = KI_{i,1} \| KI_{i,2} \| KI_{i,3} \quad (7)$$

동작을 식으로 표현하면 다음과 같다.

$$R_j = FI(L_{j-1} \oplus KO_{i,j}, KI_{i,j}) \oplus R_{j-1} \quad (8)$$

$$L_j = R_{j-1} \ (1 \le j \le 3) \quad (9)$$

FO 블록 내의 FI 블록은 [그림 4]와 같이 16비트 데이터 입력과 16비트 서브키를 필요로 하는데, 7비트 입력에 대하여 7비트 출력을 내는 S7 박스와 9비트 입력에 대하여 9비트 출력을 내는 S9 박스로 구성된다.

동작을 식으로 표현하면 아래와 같다. 여기서, ZE( ) 함수는 7비트 입력에 대하여 상위 2비트에 "00"을 추가하여 9비트 출력을 내도록 하며, TR( ) 함수는 9비트 입력에 대하여 상위 2비트를 버려서 7비트 출력을 만든다.

$$L_1 = R_0 \qquad R_1 = S9[L_0] \ ZE(R_0) \quad (10)$$

$$L_2 = R_1 \ KI_{i,j,2} \qquad R_2 = S7[L_1] \ TR(R_1) \ KI_{i,j,1} \quad (11)$$

$$L_3 = R_2 \qquad R_3 = S9[L_2] \ ZE(R_2) \quad (12)$$

$$L_4 = S7[L_3] TR(R_3) \qquad R_4 = R_3 \quad (13)$$

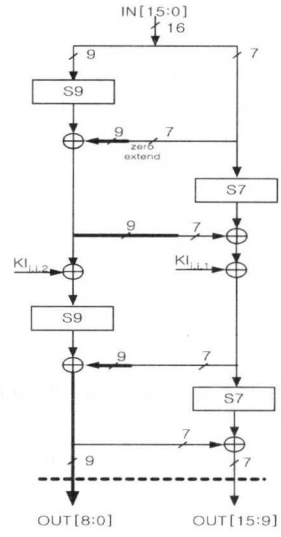


그림 4. FI 블록의 구조

KASUMI 알고리즘은 128비트 길이의 키 K를 사용하며, 각 라운드에 사용되는 키 값은 이 K값을 이용하여 [표 1]과 같이 생성된다. 이때, K는 8개의 16비트 길이의 키 K_1....K_8으로 구성되며, $K_j' = K_j \oplus C_j$ $(1 \le j \le 8)$이다. C_j는 키 스케줄 상수로 [표 2]와 같다.

표 1. 키 스케줄 표

Round	1	2	3	4
$KL_{i,1}$	K_1<<<1	K_2<<<1	K_3<<<1	K_4<<<1
$KL_{i,2}$	K_3'	K_4'	K_5'	K_6'
$KO_{i,1}$	K_2<<<5	K_3<<<5	K_4<<<5	K_5<<<5
$KO_{i,2}$	K_6<<<8	K_7<<<8	K_8<<<8	K_1<<<8
$KO_{i,3}$	K_7<<<13	K_8<<<13	K_1<<<13	K_2<<<13
$KI_{i,1}$	K_5'	K_6'	K_7'	K_8'
$KI_{i,2}$	K_4'	K_5'	K_6'	K_7'
$KI_{i,3}$	K_8'	K_1'	K_2'	K_3'

Round	5	6	7	8
$KL_{i,1}$	K_5<<<1	K_6<<<1	K_7<<<1	K_8<<<1
$KL_{i,2}$	K_7'	K_8'	K_1'	K_2'
$KO_{i,1}$	K_6<<<5	K_7<<<5	K_8<<<5	K_1<<<5
$KO_{i,2}$	K_2<<<8	K_3<<<8	K_4<<<8	K_5<<<8
$KO_{i,3}$	K_3<<<13	K_4<<<13	K_5<<<13	K_6<<<13
$KI_{i,1}$	K_1'	K_2'	K_3'	K_4'
$KI_{i,2}$	K_8'	K_1'	K_2'	K_3'
$KI_{i,3}$	K_4'	K_5'	K_6'	K_7'

표 2. 키 스케줄 상수

C_1	0x0123	C_2	0x4567
C_3	0x89AB	C_4	0xCDEF
C_5	0xFEDC	C_6	0xBA98
C_7	0x7654	C_8	0x3210

2. KASUMI 하드웨어 구현 및 특성

KASUMI 알고리즘의 하드웨어 구현은 면적과 성능에 따라서 다양한 구현이 가능하다. 특히, 라운드 구현 방법과 S-BOX 구현 방법은 하드웨어의 성능과 면적에 많은 영향을 준다.

KASUMI 하드웨어의 가장 간단한 구현 방법은 [그림 1]과 같이 8라운드를 모두 구현하는 것이다. 이러한 구조는 하드웨어를 알고리즘 그대로 구현하기 때문에 별도의 제어가 필요 없어 구현은 쉽지만, 하드웨어 면적이 크고 동작 주파수가 낮아 현실적이지 못하다. 이러한 구조에서 키 스케줄러는 모든 라운드에 대한 출력을 가지도록 설계되어야 한다.

8라운드 구조의 KASUMI 하드웨어는 [그림 5]와 같이 각 라운드 사이에 파이프라인 레지스터를 첨가하여 동작 주파수를 높일 수 있다. 이와 같은 구조는 하나의 데이터만 처리하는 경우에는 동작 주파수만 높아질 뿐 실제 초당 데이터 처리율은 높아지지 않는다. 하지만, 파이프라인 레지스터와 함께 별도의 키 스케줄러를 추가하면 파이프라인 기법을 이용하여 8개의 데이터를 병렬로 처리할 수 있어 초당 데이터 처리율을 높일 수 있다[4].

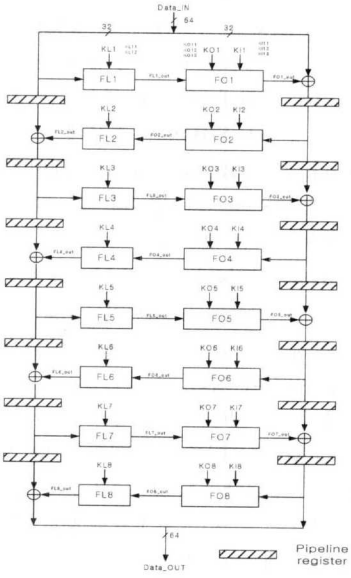


그림 5. 파이프라인된 8라운드 KASUMI 하드웨어 구조

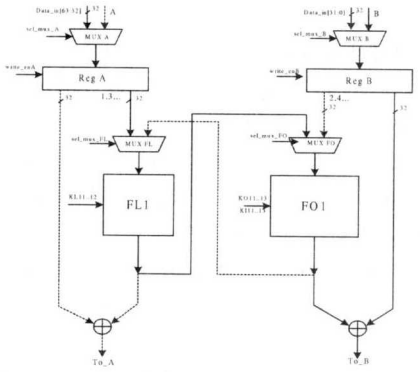


그림 6. 1라운드 KASUMI 하드웨어 구조

KASUMI 알고리즘은 FL함수와 FO함수를 각 라운드에서 반복적으로 사용하므로, 이러한 두 함수만을 구현하여 알고리즘을 수행할 수 있다. 이는 결국 1라운드만 이용하여 알고리즘을 수행하도록 구현하는 것으로, 하드웨어 면적을 줄이면서 동작주파수도 높일 수 있다. [그림 6]은 이러한 1라운드 구조의 KASUMI 하드웨어 구조를 나타낸다.

그림에서 보는 바와 같이 FL블록과 FO블록 입력단의 MUX를 제어하여 홀수번째 라운드와 짝수번째 라운드의 데이터 경로를 설정한다. 최종결과는 파이프라인된 8라운드 KASUMI 하드웨어와 마찬가지로 8 클럭 후 출력되며, 동작 주파수 또한 거의 같다. 이와 같은 구조의 또 다른 이점은 키 스케줄러 하드웨어 면적이 작다는 것이다. 아래 [그림 7]은 이러한 키 스케줄러 구조이다. 그림에서 보는 바와 같이 키 스케줄러의 출력은 한 라운드에 필요한 키 값들 만을 출력한다. KASUMI 하드웨어에서 처리하는 라운드가 진행함에 따라 키 스케줄러의 키 레지스터와 상수 레지스터가 쉬프트하게 되고, 그에 따라 각 라운드에 필요한 키 값들이 새로이 생성되게 된다.

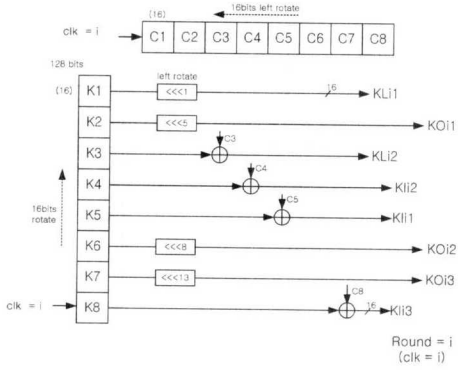
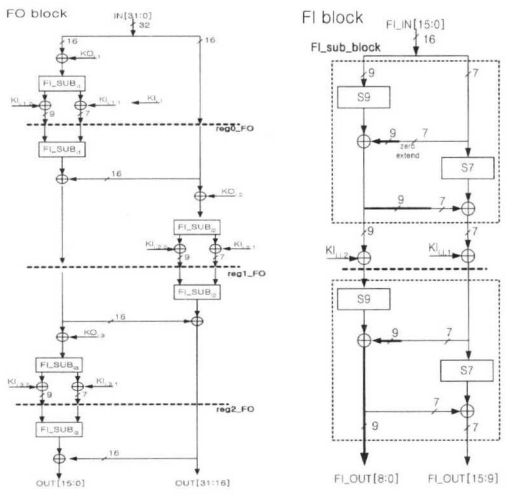


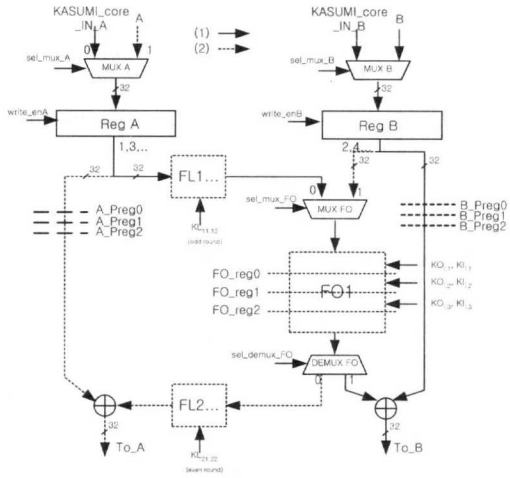
그림 7. 키 스케줄러 하드웨어 구조

KASUMI 하드웨어가 더욱 높은 동작 주파수에서 동작하여야 하는 경우에는 한 라운드를 다시 파이프라인 시켜 구현할 수 있다. 한 라운드를 다시 파이프라인 하기 위해서는 많은 연산 시간을 소요하는 FO 블록을 파이프라인 하는 것이 효율적인데, 파이프라인 사이의 연산 블록들의 연산 소요시간이 가능한 균등하게 되도록 하여야 한다.

[그림 8.a]는 FO 블록에 3단의 파이프라인 레지스터를 첨가하여 전체적으로 4단 파이프라인 구조가 되도록 구현한 FO 블록도로 점선이 파이프라인 레지스터를 나타낸다. [그림 8.b]는 파이프라인된 FI 블록이며, [그림 8.c]는 4단 파이프라인 구조의 KASUMI 하드웨어 블록도이다. [그림 8.c]에서 보는 바와 같이 FL 블록은 두 개가 사용되었는데, 이는 홀수번째 라운드와 짝수번째 라운드의 FL 블록이 충돌되는 것을 막기 위해서이다. 이와 같은 파이프라인 구조에서 파이프라인 효과에 의한 병렬처리가 가능하게 하기 위해서는 별도의 키 스케줄러가 첨가되어야 한다. 이 경우 각 키 스케줄러의 출력은 라운드 수와 파이프라인 단계에 따라 선택적으로 연산블록에 입력되어야 한다. 그런데 이러한 키 스케줄러를 가지는 KASUMI 하드웨어를 f8과 f9 알고리즘 구현에 이용하는 경우에는 KASUMI 하드웨어 동작 뿐만 아니라, f8 / f9 하드웨어 동작도 매우 복잡하게 된다. 이는 각 알고리즘의 처리 데이터 길이가 가변적인데, 이러한 입력 데이터들을 파이프라인 처리하기가 복잡하기 때문이다. 따라서 여기서는 파이프라인 기법을 단순히 동작 주파수를 높이기 위해 사용하는 것으로 한다.



(a) 파이프라인된 FO 블록 (b) 파이프라인된 FI 블록



(c) KASUMI 하드웨어 구조

그림 8. 4단 파이프라인 구조의 KASUMI 하드웨어 구조

위에서 언급하였듯이 KASUMI 하드웨어는 S-Box 구현 방법에 따라서도 성능이 크게 달라질 수 있다. KASUMI 암호 알고리즘의 S-Box 구현 방법은 크게 메모리 테이블로 구현하는 방법과 논리 게이트로 구현하는 방법이 있다. 메모리 테이블로 구현하는 방법은 다시 RAM 메모리를 이용하는 방법과 ROM 메모리를 이용하는 방법으로 구분할 수 있는데, RAM 메모리로 S-Box를 구현하면 빠른 동작 속도를 얻을 수 있지만 많은 메모리를 필요로 하는 단점이 있다. KASUMI 알고리즘의 경우 1라운드 만을 하드웨어로 구현한다고 할지라도 S7 S-Box와 S9 S-Box가 각각 6개씩 필요로 하기 때문에 많은 메모리가 소요된다. 구체적으로 한 라운드의 S9 S-Box 구현에 필요한 메모리는 2^9*9*6 = 27,648 bits이며, S7 S-Box의 경우에는 2^7*7*6 = 5,376 bits이다. 또한 RAM 메모리로 테이블을 구성하려면 하드웨어가 초기화 될 때마다 테이블 값을 다시 저장하여야 하는 번거로움이 있다. ROM 메모리로 S-Box를 구현하는 경우에는 면적이 RAM 메모리일 때보다 작으며 초기화가 필요 없는 장점이 있지만, 억세스 속도가 느려 빠른 동작 주파수를 얻을 수 없는 단점이 있다.

이에 비해 논리 게이트로 S-Box를 구현하면 동작 속도는 RAM 메모리를 사용할 때보다 느리지만, 작은 면적으로 구현할 수 있는 장점이 있다. 아래 [표 3]은 1 라운드 구조의 KASUMI 하드웨어의 S-Box를 RAM 메모리, ROM 메모리, 논리 게이트 등으로 구현했을 때 결과를 비교한 것이다.

표 3. 1 라운드 KASUMI 구현 결과 (S-Box 구현에 따라)

	Slices	Block RAM	f_{max} (MHz)
RAM 구현	271 (3%)	18 (75%)	35.62
ROM 구현	1449 (20%)	n/a	11.93
논리게이트 구현	574 (8%)	n/a	14.34

[n/a : not appliable]

표의 결과는 Xilinx FPGA XCV600E를 이용하여 구현한 결과로서, RAM으로 구현한 결과는 면적(slices)이 작으며 매우 빠른 동작 주파수를 얻을 수 있지만, 많은 메모리(전체 메모리의 75%)를 사용함을 알 수 있다. 반면에 논리게이트로 구현한 결과는 RAM으로 구현할 때보다 면적(slices)이 크며 동작 주파수도 낮지만, 메모리를 사용하지 않기 때문에 메모리를 다른 용도로 사용할 수 있는 이점이 있다. (메모리 면적까지 고려하면 사실 논리게이트로 구현한 결과의 면적이 훨씬 작다.) 그리고 ROM으로 구현한 결과는 Xilinx FPGA 특성상, ROM 테이블을 논리 게이트로 구성하기 때문에 면적과 동작 속도 모두 좋지 못한 결과를 보였다.

S-Box를 논리게이트로 구현한 경우 동작 주파수를 높이기 위해서, 위에서 언급한 파이프라인 기법을 이용할 수 있다. [표 4]는 S-Box를 논리 게이트로 구현하였을 때, 라운드 구조에 따른 하드웨어 설계 결과를 보인 것으로 4단 파이프라인 구조를 취하면 RAM 메모리로 S-Box를 구현할 때와 유사한 동작 주파수를 얻을 수 있다. 그리고 8라운드 구조의 경우 면적이 크고, 동작 주파수는 낮음을 확인할 수 있다.

표 4. 라운드 구조에 따른 구현 결과

	Slices	$Delay_{max}$ (ns)	f_{max} (MHz)
8라운드 구조	3492 (50%)	435.75	2.29
1라운드 구조	574 (8%)	69.72	14.34
1라운드 4단 파이프라인	594 (9%)	27.55	36.29

Ⅲ. 기밀성 및 무결성 알고리즘

본 절에서는 f8 기밀성 알고리즘과 f9 무결성 알고리즘에 대하여 살펴보고, 이의 하드웨어 구현에 대하여 살펴본다.

1. f8 기밀성 알고리즘

f8 기밀성 알고리즘은 KASUMI 암호 알고리즘을 사용하여, 5114비트 이하의 길이를 가지는 데이터에 대한 암호화/복호화를 수행한다. f8 기밀성 알고리즘 구조는 [그림 9]와 같으며, 입출력 파라미터는 [표 5]와 같다[1].

[그림 9]에서 보는 바와 같이, 초기 A 값을 만들기 위해 64비트 길이의 COUNT ‖ BEARER ‖ DIRECTION ‖ 00..00 값이 KASUMI 블록의 입력으로 사용되고, CK ⊕ KM(0x555...5) 키 값이 입력된다. KASUMI 암호 블록은 이들 입력에 대해 암호화 동작을 수행하여, 초기 A 값을 생성한다. 생성된 초기 A 값은 향후, f8 기밀성 알고리즘 수행 시, BLKCNT값 및 KASUMI 키 스트림 값과 exclusive OR 되어 계속 사용된다.

KASUMI 블록은 초기 A 값을 계산하는데 먼저 사용되고, 그 이후에는 키 스트림을 만드는데 사용된다. 이때, 사용되는 키 값은 CK이며, 입력값은 A ⊕ BLKCNT ⊕ KS(이전 단의 키 스트림 값)이다. BLKCNT 값은 64비트로서 키 스트림 계산이 끝나면 자동적으로 1씩 증가한다. 생성된 키 스트림 값은 입력 비트 스트림인 IBS와 exclusive 연산되어 출력 비트 스트림 (OBS)값을 생성한다.

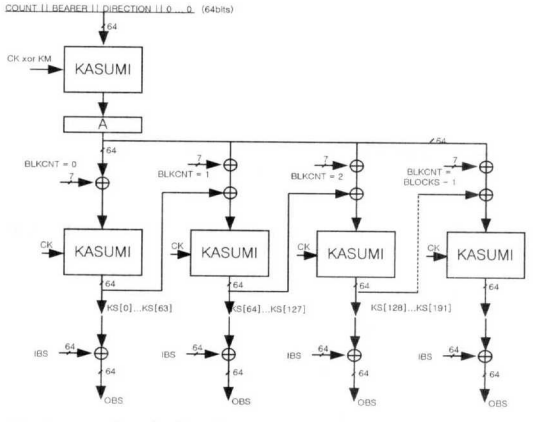


그림 9. F8 암호화 알고리즘 구조도

표 5. f8 기밀성 알고리즘의 입출력 파라미터

파라미터	크기 (bits)	기 능
COUNT	32	프레임에 의해 정의되는 32비트 값 COUNT[0] .... COUNT[31]
BEARER	5	Bearer ID값 BEARER[0] .... BEARER[4]
DIREC-TION	1	전송 방향 DIRECTION[0]
CK	128	128비트 기밀성 키 값
LENGTH		암호화/복호화 대상의 비트 길이
IBS	1 ~ 5114	입력 비트 스트림 IBS[0] .... IBS[LENGTH - 1]
OBS	1 ~ 5114	출력 비트 스트림 OBS[0] ... OBS[LENGTH - 1]

2. f9 무결성 알고리즘

f9 무결성 알고리즘도 f8 기밀성 알고리즘과 마찬가지로 KASUMI 암호 알고리즘을 사용하여 구현하며, 5114비트 이하의 길이를 가지는 데이터에 대하여 무결성 검증을 위한 MAC 값을 생성한다. f9 무결성 알고리즘 구조는 [그림 10]와 같다. 입출력 파라미터는 [표 6]와 같다[1].

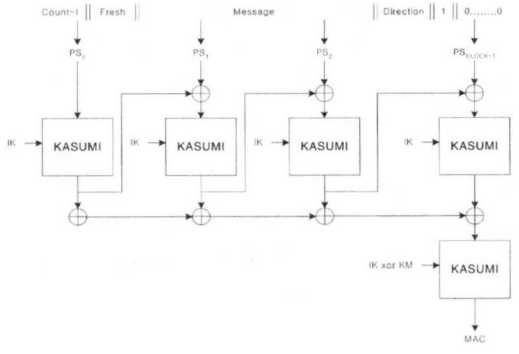


그림 10. f9 무결성 검증 알고리즘 구조도

표 6. f9 무결성 알고리즘의 입출력 파라미터

파라미터	크기 (bits)	기 능
COUNT-I	32	프레임에 의해 정의되는 32비트 값 COUNT-I[0] ... COUNT-I[31]
FRESH	32	랜덤 수 FRESH[0] .. FRESH[31]
DIREC-TION	1	전송 방향 DIRECTION[0]
IK	128	128비트 무결성 키 값
LENGTH		비트 길이
MESS-AGE	1 ~ 5114	입력 비트 스트림
MAC-I	32	메시지 인증 코드 MAC-I[0] ... MAC-I[31]

[그림 10]에서 보는 바와 같이, COUNT-I (Integrity)와 FRESH 값으로 64비트 입력을 만들어 무결성 검증 키 IK로 KASUMI 알고리즘을 수행한다. 이후 메시지들에 대하여 CBC(Cipher Block Chaining) 모드로 KASUMI 알고리즘을 수행한다. 마지막 데이터에 대해서는 DIRECTION||1||0..0을 패딩(padding)하여 64비트 입력을 생성한다. 이때, '1'까지 패딩되어 64비트가 만들어지면 '0'은 전혀 사용되지 않지만, DIRECTION 값 패딩으로 64비트 입력이 만들어지면, '1'과 63비트의 '00...0'으로 새로운 입력을 추가하여야 한다.

모든 입력 데이터에 대하여 알고리즘이 수행되면, 최종 출력 데이터와 전 단의 출력 데이터에 대하여 다시 한번 KAUSMI 알고리즘을 수행한다. 이때 입력 키 값은 IK ⊕ KM이며, 출력의 상위 32비트만을 취하여 MAC 값을 생성한다.

생성된 MAC 값은 [그림 11]과 같이 메시지에 덧붙여서 전송하고, 수신자는 메시지에 대하여 같은 방식으로 XMAC을 생성하여 전송된 MAC 값과 비교함으로써 메시지 무결성을 검증한다[3].

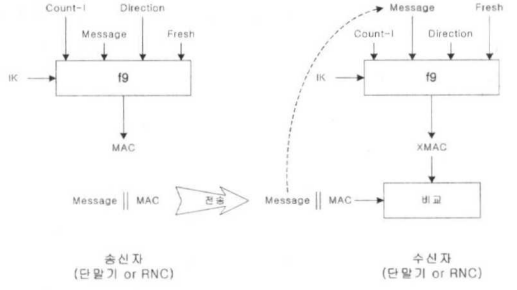


그림 11. f9 무결성 검증 절차

3. 기밀성 및 무결성 하드웨어 구현

f8 기밀성 알고리즘은 [그림 12]와 같이 KASUMI 암호 블록을 반복적으로 사용하는 구조로 구현된다. 구체적으로 살표보면, 초기값(Init)으로 COUNT||BEARER||DIRECTION||00...0 값을 입력받아 A 레지스터에 저장한 후, 초기값과 CK⊕ KM을 키 값으로 입력 받아 KASUMI 동작을 수행하여, 초기 A 값을 생성하여 다시 A 레지스터에 저장한다. 저장된 초기 A 값은 Counter 블록으로부터 '1'씩 증가되어 출력되는 BLKCNT 값과 전단계에서 생성되는 키 스트림과 XOR 연산되어 KASUMI 입력으로 사용된다. 이렇게 생성된 KASUMI 입력 값은 CK 키 값으로 암호화 되어 키 스트림으로 출력되고, KS 레지스터에 저장된다. KS 레지스터에

저장되는 키 스트림 값은 다음 KASUMI 입력값으로 사용됨과 동시에 메모리에 저장되어 있는 메시지 데이터를 암호화하는데 사용된다. 암호화된 메시지 데이터는 다시 메모리에 저장된다.

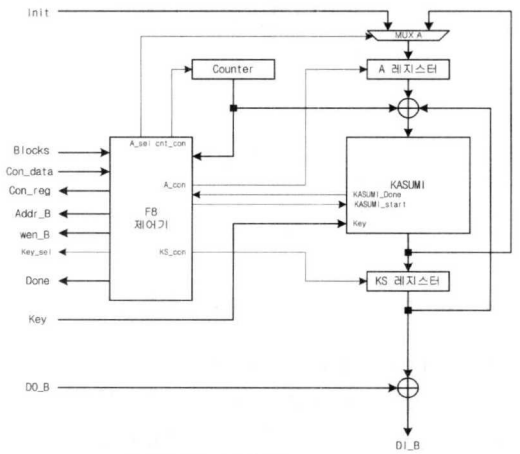


그림 12. f8 기밀성 하드웨어 구조

f9 무결성 검증 하드웨어 역시 [그림 13]과 같이 KASUMI 암호 블록을 반복적으로 사용하는 구조로 구현된다. 초기값(Init)으로 COUNT||FRESH 값을 입력받아 A 레지스터에 저장한 후, 저장된 초기값과 IK 키 값으로 KASUMI 동작을 수행한다. KASUMI 출력은 KS 레지스터에 저장되어 다음 단계의 KASUMI 입력으로 사용된다. B 레지스터는 KASUMI 출력 데이터를 계속 누적하며 저장하는데 사용된다. 초기값에 대한 KASUMI 동작이 완료되면, 이후 메모리 데이터에 대하여 같은 동작을 수행한다. 메모리로부터 전송되는 마지막 데이터에 대해서는 Init 데이터로 64비트 padding 데이터가 전송되고, MUX_A에서는 이 두 데이터를 조합하여 KASUMI 입력값을 만든다. 만약 마지막 전송 데이터 길이가 63비트인 경우에는 1||00..00 으로 구성되는 64비트 데이터로 한 번 더 KASUMI 알고리즘을 수행하며, 마지막 전송 데이터 길이가 64비트인 경우에는 DIRECTION||1||00...00으로 구성되는 64비트 데이터로 다시 한번 KASUMI 알고리즘을 수행한다. 모든 데이터에 대한 KASUMI 암호 동작이 완료되면, B 레지스터에 저장된 값과 IK ⊕ KM 키 값으로 한 번 더 KASUMI 알고리즘을 수행한 후 상위 32비트 값을 MAC 값으로 출력한다.

[그림 12]와 [그림 13]을 살펴보면, 두 하드웨어 구조가 매우 유사함을 알 수 있다. 따라서 이 두가지 하드웨어는 [그림 14]와 같이 약간의 로직을 첨가하여 하나의 하드웨어로 최적화가 가능하다. 그림에서 보듯이 최적화된 하드웨어는 두 가지 동작을 위한 데이터 패스가 모두 구현되어 있다. 이들 동작은 제어 레지스터의 값에 따라 구분되어 수행된다.

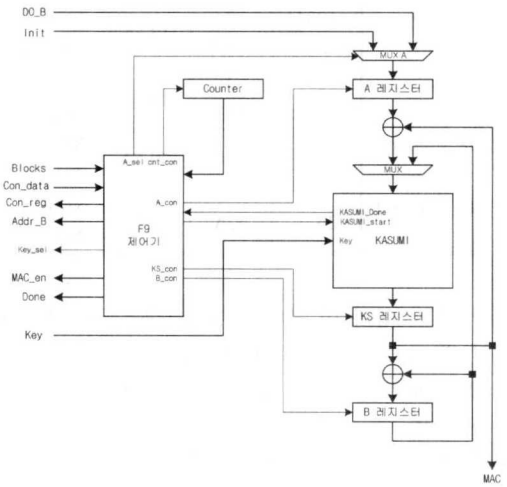


그림 13. f9 무결성 하드웨어 구조

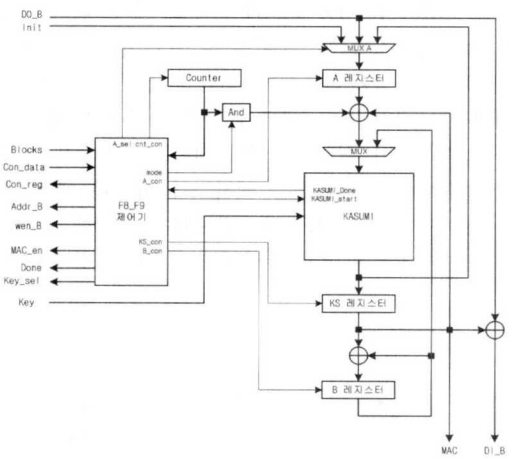


그림 14. f8 기밀성 및 f9 무결성 하드웨어 구조

[표 7]은 f8 기밀성 하드웨어와 f9 무결성 하드웨어를 각각 구현하였을 때와 두 하드웨어를 최적화하여 구현하였을 때 결과를 비교한 것이다. 표에서 보듯이 최적화하여 구현한 결과와 각각 구현한 결과가 면적과 성능 면에서 별 차이가 없음을 알 수 있다. (KASUMI 하드웨어는 1라운드 구조가 사용되었으며, Xilinx FPGA XCV600e에서 구현되었다.)

[표 7]에서 f8_2_rounds_comb와 f9_2_rounds_comb는 논문 [4]에서의 구현 결과이다. 이는 2라운드로 구현된 KASUMI 블록 2개를 이용하여 f8 기

표 7. f8 / f9 하드웨어 구현 결과 비교

구분	Slices	Block RAM	f_{max} (MHz)	Device
f8 구현	995 (14.3%)	2 (8%)	13.62	XCV 600E
f9 구현	1034 (14.9%)	2 (8%)	13.56	XCV 600E
f8 / f9 구현	1087 (15%)	2 (8%)	13.15	XCV 600E
f8_2_rounds _comb[4]	2781	N/A	20.52	XCV 300E
f9_2_rounds _comb[4]	2671	N/A	20.68	XCV 300E

밀성 알고리즘과 f9 무결성 알고리즘을 구현한 결과로서 본 논문에서의 결과보다 동작 주파수는 높지만, 면적(slices)이 훨씬 큼을 알 수 있다. 동작 주파수의 차이는 구현된 FPGA 디바이스 차이와 KASUMI 하드웨어와 f8, f9 하드웨어 최적화를 위하여 추가된 로직의 데이터 패스 딜레이 때문이다.

논문 [4]와 본 논문에서의 각 하드웨어는 KASUMI와 f8, f9 하드웨어 구조 뿐 아니라 입출력 인터페이스에서도 차이를 보인다. 선행 논문에서는 f8 또는 f9 하드웨어 동작을 상위 시스템에서 제어하고 키와 기타 제어 값들을 동작에 맞게 적절히 입력하여야 한다. 본 논문에서는 상위 시스템과의 입출력 인터페이스는 레지스터 파일과 메모리를 통하여 구현되며, f8 / f9 하드웨어 동작에 필요한 입력들은 내부 제어기에 의해 결정된다(그림 15).

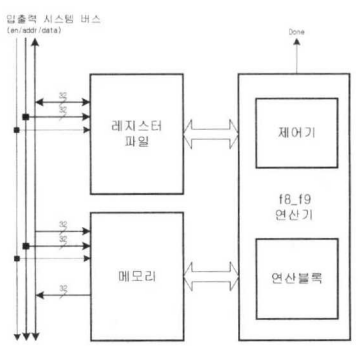


그림 15. f8 / f9 하드웨어 상위 구조

[그림 15]의 메모리는 입출력 데이터를 저장하기 위해 사용된다. 메모리는 두 알고리즘의 최대 처리 데이터 길이인 5114비트를 저장할 수 있도록 256x32 RAM 메모리가 사용되었다. 메모리에 저장된 데이터는 f8 / f9 제어기에 의해 f8 / f9 연산기로 불려져 처리되며, 처리된 데이터는 다시 메모리에 저장되거나 MAC 레지스터에 저장된다.

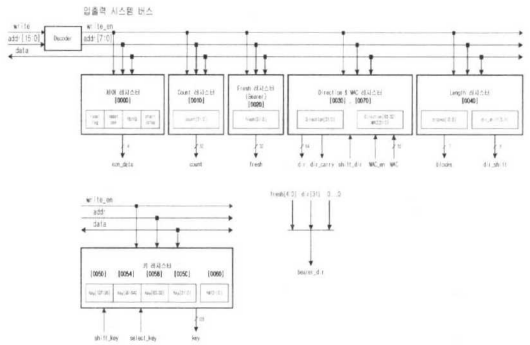


그림 16. f8 / f9 하드웨어 레지스터 파일 구조

[그림 16]의 레지스터 파일은 f8 기밀성 알고리즘과 f9 무결성 알고리즘의 파라미터를 저장 및 최종 MAC 데이터 저장하기 위하여 사용된다. 구체적으로, 제어 레지스터, COUNT 레지스터, FRESH 레지스터, DIRCTION 레지스터, LENGTH 레지스터, 키 레지스터로 구성된다. 이들 레지스터 중 FRESH 레지스터는 f8 기밀성 알고리즘에서는 하위 5비트에 BEARER 값을 저장한다. DIRECTION 레지스터는 f8 기밀성 알고리즘과 f9 무결성 알고리즘에서 데이터 전송 방향을 나타내는 정보와 메시지 padding 데이터를 저장하는데 사용되며, f9 무결성 알고리즘 동작이 완료되면 MAC 값을 저장하는데 사용된다. f8 / f9 하드웨어는 이러한 레지스터 파일 값을 참조하여 동작에 필요한 적절한 입력값을 내부적으로 생성하며 동작한다.

4. 시연 시스템 구현 및 분석

설계된 f8 기밀성 및 f9 무결성 하드웨어가 외부와 통신하기 위해서는 적절한 인터페이스 블록을 가져야 한다. 본 설계에서는 현재 PC 상에서 가장 많이 사용되고 있는 주변 장치용 버스인 PCI(Peripheral Component Interconnect)를 사용하였다. [그림 17]은 PCI 버스 인터페이스를 가지는 f8 기밀성 및 f9 무결성 하드웨어 블록도이다. 그림에서 보는 바와 같이 f8 / f9 하드웨어는 내부 시스템 버스로 PCI 인터페이스 블록과 연결되어 있으며, FPGA의 용량에 따라서 최대 8개의 f8 / f9 연산 모듈을 구현할 수 있도록 설계되었다.

위와 같은 하드웨어는 [그림 18]과 같은 PCI 보드로 구현되어 테스트 되었다. 사용한 FPGA 칩은 Xilinx XCV 600E이며, PCI 동작 주파수인 33MHz

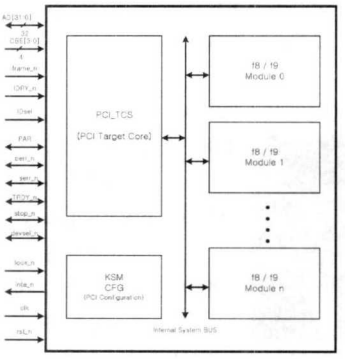


그림 17. PCI 인터페이스를 가지는 f8 / f9 하드웨어 블록

에서 동작하기 위하여 4단 파이프라인 구조의 KASUMI 블록이 사용되었으며, 시스템의 채널 용량 증가를 위하여 하나의 칩 안에 6개의 f8 / f9 하드웨어 모듈이 구현되었다. 여기서 33MHz의 동작 주파수는 [표 3]에 보인 바와 같은 램 메모리를 사용한 KASUMI 블록을 이용하여도 만족시킬 수 있지만, 이 경우 사용된 FPGA 내부 메모리 용량을 초과하여 이러한 구조는 취할 수 없었다. 구현된 시스템 성능은 [표 8]과 같다.

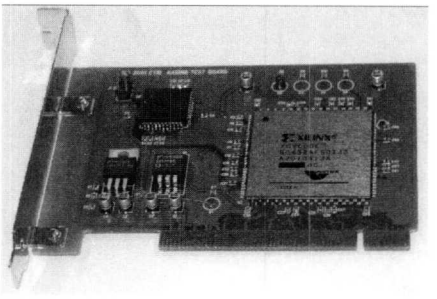


그림 18. f8 / f9 하드웨어 시연 보드

표에서 보는 바와 같이 구현된 시스템의 하나의 f8 / f9 하드웨어 모듈의 성능은 55 Mbps 정도이며, 6개의 연산 모듈을 동시에 사용하는 경우에는 각각의 병렬성이 보장되기 때문에 최대 330 Mbps의 성능을 얻을 수 있다. 표에서 비교된 두 하드웨어 결과는 모두 Xilinx XCV 600E으로 구현된 결과로서, 비교된 Marinis[4]의 결과는 메모리 한계로 더 이상의 연산 모듈을 수용할 수 없다. 따라서 동일한 FPGA를 사용한 하드웨어 구현을 고려하면, 최대 성능 면에서 본 논문의 결과가 대략 5배정도 우수함을 알 수 있다. 또한 본 논문의 결과는 하나의 하드웨어 모듈이 f8 알고리즘과 f9 알고리즘을 모두 처리할 수 있으며, 실용적인 외부 인터페이스 구조를 가지는 장점이 있다. 만약 본 논문의 결과를 큰 메모리 자원을 가지는 칩으로 구현하면, KASUMI 블록의 S-Box를 램 메모리로 구현이 가능하며, 이러한 경우 하나의 연산 모듈만으로 220Mbps 정도의 성능을 얻을 수 있다.

표 8. f8 / f9 하드웨어 시연 보드 구현 결과

Item	Our	Marinis[4]
Slices	6862 / 6912(99%)	1563(f8) 1560(f9)
Block RAM	12 / 24 (50%)	48(f8) 48(f9)
최대 동작 주파수	34.23 MHz	33.14(f8) 33.52(f9)
동작 주파수	33 MHz	N/A
성능 (모듈 1개)	55 Mbps	70.7 Mbps
최대 성능 (모듈 6개)	330 Mbps	N/A

Ⅳ. 결 론

본 논문에서는 IMT-2000 시스템을 위한 기밀성 및 무결성 하드웨어를 구현하였다. 이를 위해 두 알고리즘의 근간이 되는 KASUMI 암호 알고리즘을 다양한 구조로 구현하여 검증하였으며, 이 결과를 바탕으로 기밀성 및 무결성 하드웨어를 구현하여 검증하였다. 구현된 하드웨어는 4단 파이프라인 구조의 KASUMI 블록을 이용하고 있으며, 기밀성 알고리즘과 무결성 알고리즘을 모두 수행할 수 있다. 또한 PCI 인터페이스를 내장하여 실제 시스템으로의 응용이 용이하도록 하였다. FPGA 시연 보드로 구현된 시스템은 33MHz 동작주파수에서 최대 330Mbps의 성능을 보인다. 이는 더 큰 자원을 가지는 FPGA로 구현하거나 ASIC으로 구현할 시 높은 성능을 보장할 것이며, 이를 RNC 시스템에 적용할 경우 시스템 성능 개선에 큰 도움이 되리라 판단된다.

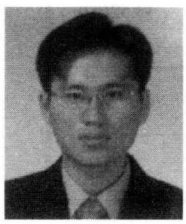
참 고 문 헌

[1] ETSI/SAGE. Specification of the 3GPP Confidentiality and Integrity Algorithms Document 1: f8 and f9 Specification, Sept. 2000

[2] ETSI/SAGE. Specification of the 3GPP Confid-

entiality and Integrity Algorithms Document 2: KASUMI Specification, Dec. 1999.
[3] 3GPP TS 33.102 v3.7.0 , Dec. 2000.
[4] Kostas Marinis, Nikos K. Moshopoulos, "On the Hardware Implementatio of the 3GPP Confidentioality and Intergirty Algorithms", ISC 2001, LNCS 2200, pp. 248-265, 2001
[5] Mitsuru Matsui, "New block encryption algorithm MISTY", Fast Software Encryption 97, LNCS 1267, pp. 54-68, 1997

최 용 제(Yong-Je Choi) 정회원

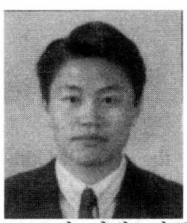


1997년 전남대학교 전자공학과 공학사
1999년 전남대학교 전자공학과 공학석사
1999년~현재 한국전자통신연구원 정보보호연구본부 정보보호기반연구팀, 연구원
<주관심 분야> 타원곡선 암호모듈 설계, 암호프로세서 설계, 정보보호

김 호 원(Ho-Won Kim) 정회원

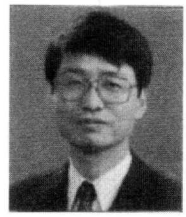
1993년 경북대학교 전자공학과 공학사
1995년 포항공과대학교 전자전기공학과 공학석사
1999년 포항공과대학교 전자전기공학과 공학박사
1998년~현재 : 한국전자통신연구원 정보보호연구본부 정보보호기반연구팀, 선임연구원
<주관심 분야> 타원곡선 암호모듈 설계, 암호프로세서 설계, 정보보호

김 무 섭(Moo-Seop Kim) 정회원



1995년 금오공과대학교 전자공학과 공학사
1998년 경북대학교 전자전기공학과 공학석사
1998년~1999 LG종합기술원
1999년~현재 : 한국전자통신연구원 정보보호기술연구본부 정보보호기반연구팀, 연구원
<주관심 분야> RSA 암호모듈 설계, 암호프로세서 설계, 정보보호

정 교 일(Kyo-Il Chung) 정회원



1981년 한양대학교 전자공학과 공학사
1983년 한양대학교 산업대학원 전자계산학과 공학석사
1997년 한양대학교 대학원 전자공학과 공학박사
1995년~1997년 : 한국정보통신기술협회 전파통신분과위원회 연구위원
1981년~현재 : 한국전자통신연구원 정보보호연구본부 정보보호기반연구부 부장/책임연구원
<주관심 분야> IC카드 설계, 정보보호 시스템