https://doi.org/10.7840/kics.2017.42.2.316

초경량 암호 PRESENT의 소프트웨어 구현 시 처리량 향상에 대한 연구

박 원 규[◆], G. P. Cebrián[*], 김 성 준[**], 이 강 현[***], 임 대 운[**], 유 기 순[°]

A Study on the Throughput Enhancement in Software Implementation of Ultra Light-Weight Cryptography PRESENT

Won-kyu Park[◆], G. P. Cebrián[*], Sung-joon Kim[**], Kang-hyun Lee[***], Dae-woon Lim[**], Ki-soon Yu[°]

요 약

본 논문은 경량 블록암호 알고리즘인 PRESENT를 소프트웨어로 구현 시 단위 시간 당 암호화 처리량을 증가시키는 기법을 제안한다. PRESENT의 각 라운드는 라운드 키 첨가, 치환, 전치 과정으로 구성되어 있으며, 이를 31회 반복 수행한다. Bo Zhu는 효율적 연산을 위해 치환과 전치 과정을 통합하여 연산하는 기법을 제안하였고, 치환과 전치를 따로 수행하는 기존 기법에 비해 암호화 처리량을 약 2.6배 증가 시켰다. 본 논문에서 제안한 기법은 Bo Zhu가 제안한 기법에서 특정 비트를 선택하기 위한 연산을 제거함으로써 암호화 성능을 개선하였다. Bo Zhu의 기법에 비해 메모리 사용량은 증가하지만, 암호화 처리량을 최대 약 1.6배 증가 시켰다.

Key Words : PRESENT, Lightweight Encryption Algorithm, IoT Security

ABSTRACT

This paper suggests an efficient software implementation of lightweight encryption algorithm PRESENT which supports for secret key lengths of 80-bits. Each round of PRESENT is composed of the round key addition, substitution, and permutation and is repeated 31 times. Bo Zhu suggested combined substitution and permutation for efficient operation so that encryption throughput has been increased 2.6 times than processing substitution and permutation at separate times. The scheme that suggested in this paper improved the scheme of Bo Zhu to reduce the number of operation for the round key addition, substitution, and permutation. The scheme that suggested in this paper has increased encryption throughput up to 1.6 times than the scheme of Bo Zhu but memory usage has been increased.

※ 본 논문은 중소기업청에서 지원하는 2016년도 산학연협력 기술개발사업(과제번호 C0398312)의 연구수행으로 인한 결과물임을 밝힙니다.

◆ First author : Department of Information Communication Engineering, Dongguk University, wku0905@gmail.com, 학생회원

° Corresponding author : Department of Information Communication Engineering, Dongguk University, ykscj39@naver.com, 학생회원

* Department of Telecommunication Engineering, University Autónoma de Barcelona, guillermo.pallares@e-campus.uab.cat

** Department of Information Communication Engineering, Dongguk University, happygrounds@naver.com, daewoonlim@gmail.com, 종신회원

*** Double H, R&D Center, vincentlee@lumidiet.com, 정회원

논문번호 : KICS2016-12-402, Received December 23, 2016; Revised January 25 2017; Accepted February 1, 2017

Ⅰ. 서 론

가전, 의료, 교통 분야 등 개인의 생활에 밀접한 다양한 산업분야에 사물인터넷(IoT)이 적용되고 있다.[1] 이에 따라 IoT 사용 시 발생할 수 있는 개인 정보의 유출 및 위/변조로 인한 보안문제가 대두되고 있다, 이러한 보안문제를 해결하기 위해 IoT에 메시지 인증, 암호화 등의 보안기술 적용이 필요하다. 하지만, 기존 암호 기술을 연산 능력과 전원 등의 자원이 제한적인 IoT기기에 그대로 적용하기에는 어려움이 있어, IoT 기기에 적합한 경량 암호 기술이 등장하게 되었다. LEA, HIGHT, PRSENT, mCRYPTON, CLEFIA 등이 대표적인 경량 암호 알고리즘이다.[2]

LEA는 블록 크기가 128-bit이고, 키 크기가 128/192/256-bit인 국내 표준 경량 암호 알고리즘으로 S-Box를 사용 하지 않고, Addition Rotation Xor (ARX) 연산만으로 구성되어 동작한다.[3]

HIGHT는 2005년 국내에서 개발한 블록 암호다. 8-bit 단위의 산술연산만으로 구성되었다. 블록 크기는 64-bit이고, 키 크기는 128-bit이다.[4]

CLEFIA는 SONY에서 제안한 블록 암호로 블록 크기는 128-bit이고, 크기가 128/192/256-bit 키를 사용한다. Feistel 구조로 설계되었으며, 키 크기에 따라 라운드 수가 바뀐다.[5]

mCRYPTON는 Crypton의 축소 버전으로 2005년 제안되었다. 블록의 크기가 64-bit이고, 키 크기는 96/128-bit로 12 라운드로 구성되었다.[6]

PRSENT는 Bodganov 등이 제안한 AES 기반의 블록 암호로, [7]의 결과에 따르면, PRSENT는 Arduino, Tmote, RP2에서 암호화 및 복호화의 수행 속도가 다른 경량 암호 알고리즘에 비해 저조한 성능을 보인다. 이에 본 논문에서는 소프트웨어 관점에서 PRESENT를 효율적으로 구현하여 수행속도를 향상시킬 수 있는 최적화 기법을 제안한다.

Ⅱ. 선행 연구

2.1 PRESENT[8]

PRESENT 알고리즘은 2007년 제안된 SPN (Substitution Permutation Network) 구조의 경량 블록 암호이다. 블록의 크기는 64-bit이고, 크기가 80-bit와 128-bit인 비밀키를 사용한다. <그림 1>과 같이 PRESENT는 31 라운드의 연산을 거쳐 암호문을 출력한다. 각 라운드는 키 스케줄링을 통해 생성한 64-bit

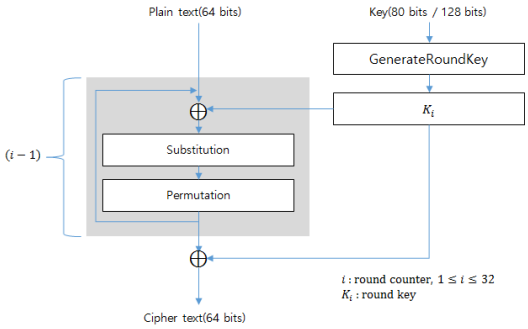

그림 1. PRESENT 알고리즘
Fig. 1. Algorithmic description of PRESENT

크기의 라운드 키 K_i를 사용한다. 마지막 라운드에서 얻은 결과 값과 XOR 연산을 한 번 더 수행하므로 총 32개의 라운드 키가 필요하다.

$$K_i = k_{63} \cdots k_0 (1 \le i \le 32)$$

키 스케줄링 방법은 다음과 같다. 첫 번째, 주어진 비밀키 $K = (k_{79}k_{78} \cdots k_1k_0)$를 61-bit 왼쪽으로 자리 순환 자리 이동(Cyclic shift)한다. 두 번째, 첫 번째 결과 값에 왼쪽 4-bit를 <표 1>과 같은 PRESENT의 S-box의 입력 값으로 한 치환한 값으로 갱신한다. 세 번째, 두 번째 결과 값에 19번~15번째 자리의 값과 round counter i를 XOR 연산한 값을 라운드 키로 사용한다.

$$1.\ [k_{79}k_{78}...k_1k_0] = [k_{18}k_{17}...k_{20}k_{19}]$$
$$2.\ [k_{79}k_{78}k_{77}k_{76}] = S[k_{79}k_{78}k_{77}k_{76}]$$
$$3.\ [k_{19}k_{18}k_{17}k_{16}k_{15}] = [k_{19}k_{18}k_{17}k_{16}k_{15}] \oplus i$$

각 라운드의 결과 값을 다음 라운드의 입력 값으로 사용하며 이를 $STATE_i = (s_0s_1 \cdots s_{63})$라하고 그 크기는 64-bit 이다.

각 라운드의 수행과정을 살펴보면, 첫 번째, 현재 $STATE_i$와 라운드 키 K_i를 XOR 연산을 수행한다. 두 번째, 첫 번째 수행결과를 4-bit씩 다음 S-Box를 사용해 치환한다. 세 번째, 두 번째 수행결과를 <표 2>와 같은 P-Box에 맞추어 각 비트 값을 전치한다. 위 과정을 모두 수행한 결과가 해당 라운드의 $STATE_i$이다.

표 1. PRESENT의 S-Box 테이블
Table 1. S-Box table of PRESENT

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

표 2. PRESENT의 P-Box 테이블
Table 2. P-Box table of PRESENT

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P(j)$	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
j	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P(j)$	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
j	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P(j)$	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
j	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P(j)$	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

2.2 Bo Zhu의 PRESENT[9]

이 기법은 PRESENT 알고리즘의 치환과 전치의 수행시간을 줄이기 위하여 S-Box와 P-Box를 합친 형태의 SP-Box를 만든 것이다. SP-Box를 만들기 위해 우선, S-Box 2개를 연접하여 입력과 출력의 크기가 8-bit인 S-Box로 확장한다. 이렇게 만들어진 확장된 S-Box의 크기는 16에서 256으로 늘어난다.

이 출력 값을 순서에 따라 2-bit씩 연접하여 전치를 수행한 8-bit의 결과 값으로 만든다. 이를 위해 다음 <그림 2>와 같이 4개의 서로 다른 SP-Box가 필요하다.

이전 라운드의 출력 $STATE_{i-1} = (s_{i-1}^0 \cdots s_{i-1}^7)$

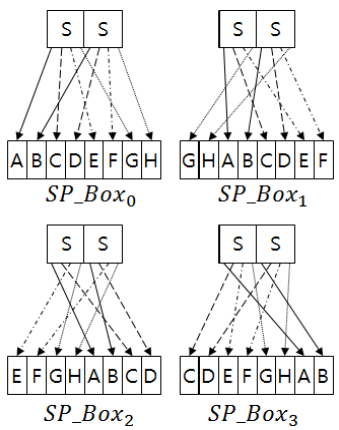

그림 2. Bo Zhu의 기법에서 SP_Box 테이블 생성
Fig. 2. Generation of SP_Box table in Bo Zhu's scheme

를 8-bit씩 나누어 해당 라운드의 입력 값으로 사용한다. s_i^0와 s_i^1의 전치 결과는 S-Box 결과의 첫 번째 값들만 연접하여 얻을 수 있으므로 다음과 같이 만들어진다.

$$\begin{aligned} s_i^0 = &\left(\text{SP\_Box}_{\ 0}\left(s_{i-1}^0\right) \odot 0\text{xC0}\right) \\ &\otimes \left(\text{SP\_Box}_{\ 1}\left(s_{i-1}^1\right) \odot 0\text{x30}\right) \\ &\otimes \left(\text{SP\_Box}_{\ 2}\left(s_{i-1}^2\right) \odot 0\text{x0C}\right) \\ &\otimes \left(\text{SP\_Box}_{\ 3}\left(s_{i-1}^3\right) \odot 0\text{x03}\right) \end{aligned}$$

$s_i^2, s_i^3 \cdots s_i^7$ 역시 위와 같이 전치를 고려한 SP-Box의 조합으로 결과 값을 얻는다.

Ⅲ. 제안 기법 구현 및 성능 비교

3.1 제안 기법 1

앞 절에서 소개한 Bo Zhu의 SP-Box 생성 방법을 바꾼 것이다. 기존의 생성 방법에서는 S-Box의 같은 위치 결과 값을 4개의 SP-Box에 나누어 저장을 하였다면, 제안하는 기법은 <그림 3>처럼 같은 위치의 결과 값을 하나의 SP-Box에 모두 저장하는 것이다.

해당 기법으로 만들어진 SP-Box는 $\text{SP_Box}_{\ 0}$만을 사용하여 s_i^0와 s_i^1을 계산할 수 있다.

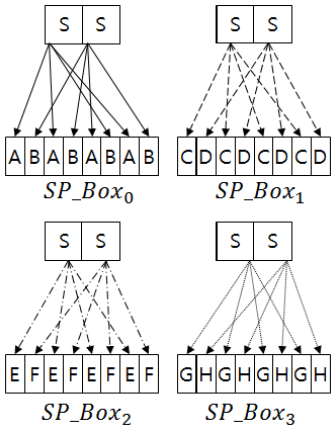

그림 3. 제안 기법 1의 SP_Box 생성법
Fig. 3. Generation of SP_Box table in Proposed scheme 1

$$
\begin{aligned}
s_i^0 = &\left(\mathrm{SP\_Box}\ {}_0\left(s_{i-1}^0\right) \odot 0\mathrm{xC0}\right) \\
&\otimes \left(\mathrm{SP\_Box}\ {}_0\left(s_{i-1}^1\right) \odot 0\mathrm{x30}\right) \\
&\otimes \left(\mathrm{SP\_Box}\ {}_0\left(s_{i-1}^2\right) \odot 0\mathrm{x0C}\right) \\
&\otimes \left(\mathrm{SP\_Box}\ {}_0\left(s_{i-1}^3\right) \odot 0\mathrm{x03}\right)
\end{aligned}
$$

$s_i^2, s_i^3 \cdots s_i^7$ 역시 위와 같은 방식으로 계산한다.

3.2 제안 기법 2

이 기법은 같은 배열에 여러 차례 접근하는 방식과 서로 다른 배열에 접근하는 방식의 수행시간 차이를 확인하기 위한 것이다. 이를 위해 3.1에서 제안한 기법을 바탕으로 동일한 SP-Box를 각각 4개씩 총 16개를 생성하였다. s_i^0의 계산 방법은 다음과 같다.

$$
\begin{aligned}
s_i^0 = &\left(\mathrm{SP\_Box}\ {}_0^0\left(s_{i-1}^0\right) \odot 0\mathrm{xC0}\right) \\
&\otimes \left(\mathrm{SP\_Box}\ {}_1^0\left(s_{i-1}^1\right) \odot 0\mathrm{x30}\right) \\
&\otimes \left(\mathrm{SP\_Box}\ {}_2^0\left(s_{i-1}^2\right) \odot 0\mathrm{x0C}\right) \\
&\otimes \left(\mathrm{SP\_Box}\ {}_3^0\left(s_{i-1}^3\right) \odot 0\mathrm{x03}\right)
\end{aligned}
$$

3.3 제안 기법 3

$STATE_i$ 계산 시 수행하는 AND 연산을 없앰으로써 수행시간을 단축하는 기법이다.

<그림 4>와 같이 이 기법에서 SP-Box는 Bo Zhu의 기법과 동일하게 S-Box의 같은 위치 결과 값을 4개의 SP-Box에 나누어 저장을 한다. 단, 해당 위치의 2-bit를 제외한 나머지를 0으로 채워 AND 연산을 없애는 것이다. 그러나 8-bit를 2개씩 저장하기 위해 4개의 SP-Box가 필요 하므로 제안 기법 2와 같이 16개의 SP-Box가 만들어진다.

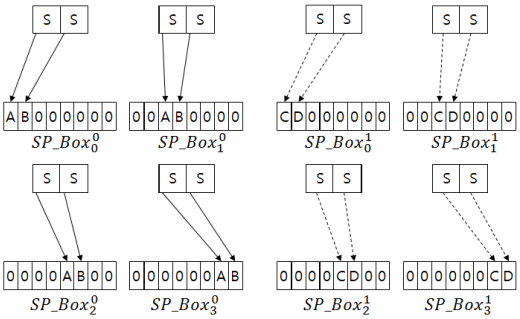

그림 4. 제안 기법 3의 SP_Box 생성법
Fig. 4. Generation of SP_Box table in Proposed scheme 3

이 기법의 SP-Box를 이용한 s_i^0의 계산 방식은 다음과 같다.

$$
\begin{aligned}
s_i^0 = &\ \mathrm{SP\_Box}\ {}_0^0\left(s_{i-1}^0\right) \\
&\otimes \mathrm{SP\_Box}\ {}_1^0\left(s_{i-1}^1\right) \\
&\otimes \mathrm{SP\_Box}\ {}_2^0\left(s_{i-1}^2\right) \\
&\otimes \mathrm{SP\_Box}\ {}_3^0\left(s_{i-1}^3\right)
\end{aligned}
$$

3.4 제안 기법 4

소프트웨어 관점에서 볼 때 암호화 수행속도를 높이기 위해 프로세서의 연산 단위와 암호 알고리즘의 연산 단위를 동일하게 하는 경우 연산 단위를 맞추기 위한 연산이 줄어 연산 복잡도를 최소화 할 수 있다.[10] 이에 32-bit 프로세서 플랫폼에 맞추기 위하여 해당 기법은 제안한 3.3 기법에서 연산 단위를 8-bit에서 32-bit로 늘린 것이다. 이 기법은 i 라운드에서 $STATE_i$와 K_i의 XOR 연산이 8회에서 2회로 줄어들고, 전체 암호 과정에서 볼 때, $STATE_i$와 K_i의 XOR 연산 횟수가 총 256회에서 총 64회로 줄어든다.

3.5 제안한 PRESENT의 성능 비교

제안한 PRESENT 암호화 기법의 수행속도 확인을 위하여 <표 3>의 환경에서 32-bit와 64-bit 플랫폼에서 기법별로 각각 암호화를 50,000,000번 수행한 결과를 평균하여 제안기법을 평가하였다.

<표 4>에서 보여주듯이 측정 결과 제안 기법 2가 제안 기법 1 보다 암호 수행속도가 빠르다. 이를 볼 때 SP-Box의 처리 결과를 얻기 위하여 동일 테이블에 여러 차례 접근하는 것보다 서로 다른 테이블에 접근하는 것이 효율적이다. 하지만, <표 5>에서 볼 수 있듯이 SP-Box의 메모리 사용량이 Bo Zhu의 기법보다 4배 증가한다.

표 3. 암호화 처리량 측정을 위한 시험 환경
Table 3. Test Environment for encryption throughput

구분	사양
CPU	Intel(R) Core(TM) i5-3470 CPU @ 3.20GHz
RAM	4GB
Hard Disk	500GB
OS	Window 10 (x64)
Compiler	Visual studio 2015(19.00.24213.1)

표 4. 암호화 처리량 측정 결과
Table 4. Test Results of encryption throughput

구분		암호 처리량 (Mbyte/second)	
		x64(64bit)	x86(32bit)
전치와 치환 분리		9.5	4.5
전치와 치환을 통합	Bo Zhu	20.8	11.8
	제안 기법 1	20.6	11.6
	제안 기법 2	21.8	11.8
	제안 기법 3	24.7	14.4
	제안 기법 4	24.9	19.7

표 5. 제안 기법별 SP-Box 메모리 사용량 비교
Table 5. Comparison of memory use for SP-Box

구분	SP-Box 메모리(Kbyte)
Bo Zhu	1
제안 기법 1	1
제안 기법 2	4
제안 기법 3	4
제안 기법 4	32

제안 기법 3은 SP-Box에서 필요한 비트를 선택하는 과정을 없앰으로 Bo Zhu의 기법보다 암호화 수행속도가 x86에서 약 1.2배 향상되었다.

암호화의 연산 단위를 32-bit로 늘린 제안 기법 4는 SP-Box의 메모리 사용량이 32Kbyte로 늘어나며, 암호를 위한 연산의 수행 횟수가 줄어들어 암호화의 수행속도는 x86에서 약 1.6배 증가하였다.

<표 6>은 메모리 사용량 측정을 위한 테스트 환경이며, <표 7>은 각 기법별 프로그램의 메모리 사용량을 나타낸다. <표 6>에서 Code는 코드가 차지하는 크기를 의미하고, RO는 Read Only 데이터로 상수 형태의 전역 변수, RW는 Read Write 데이터로 초기 값을 가지는 전역 변수, ZI Zero Initialized는 초기 값이 0

표 6. 메모리 사용량 측정을 위한 시험 환경
Table. 6. Test environment for memory use

구분	사양
Board Model	nRF51822_xxac
CPU	ARM cortex-m0(32bit, 16MHZ)
RAM	4GB
Flash	256Kbyte
Compiler	Keil v5

표 7. 메모리 사용량 측정 결과
Table. 7. Test results of memory use

구분	Code (byte)	RO (byte)	RW (byte)	ZI (byte)	Flash (byte)	RAM (byte)
Bo Zhu	2054	1248	16	2368	3318	2384
제안 기법 1	2294	224	16	2368	3184	2384
제안 기법 2	4332	224	16	2368	4572	2384
제안 기법 3	3052	224	16	2368	3292	2384
제안 기법 4	4476	224	4	3628	4704	3632

인 전역 변수를 의미한다. Flash는 Code, RO, RW를 더한 것이고, RAM은 RW와 ZI를 더한 것이다.

제안 기법 2와 제안 기법 4는 SP_Box를 구성하기 위한 메모리 사용량이 증가하면서 프로그램을 구성하는 메모리 사용량도 증가하였다. 하지만, 제안 기법 3의 경우 SP_Box를 구성하기 위한 메모리 사용량은 증가하였지만 프로그램을 구성하는 전체 메모리 사용량은 줄어 IoT 단말기에서 PRSENT 암호 알고리즘을 구현하는데 있어 적합한 것으로 보인다.

Ⅳ. 결 론

기존 PRESENT는 치환과 전치 연산을 위해 분리된 S-Box와 P-Box 만들어 처리하였다. Bo Zhu는 분리된 S-Box와 P-Box를 통합한 형태의 SP-Box를 만들고, 치환과 전치 연산 결과를 얻기 위해 SP-Box에서 필요한 비트를 선택하고 조합하여 사용하였다. 이로 인해 암호 수행속도가 기존 PRESENT보다 약 2.6배 증가 하였다.

본 논문에서 제안한 기법은 Bo Zhu의 기법에서 비트 선택의 과정을 없애고자 4개의 SP-Box를 16개로 분리하여 만들었다. SP-Box의 메모리 사용량은 4배

많아지지만 수행속도가 1.2배 증가하는 결과를 얻었다. 결과적으로, 메모리 사용량은 증가했지만 연산의 횟수를 줄임으로써 암호화의 수행속도를 높일 수 있었다. 본 연구에서 제안된 기법은 전력 소모량과 메모리 사용량 중 전력의 소모량을 최소화 하는 것이 우선적으로 필요한 IoT 환경에 적합한 대안이 될 수 있다

본 논문은 PRESENT 암호화 알고리즘의 수행 속도를 높이는 방법을 제한하였고, 향 후 PRESENT의 복호화 수행속도를 높일 수 있는 효율적인 기법에 대한 연구가 필요하다.

References

[1] D. Kim, S. Yuk, and Y. Lee, “Security for (IoT) Service,” *J. KICS*, vol. 30, no. 8, pp. 53-59, Jun. 2013.

[2] Y. Won, “IoT(Internet of Things) information security technology development direction,” *J. KICS*, vol. 32, no. 1, pp. 24-27, Dec. 2014.

[3] D. Hong, J.-K. Lee, D.-C. Kim, D. Kwon, K. H. Ryu, and D.-G. Lee, “LEA: A 128-Bit block cipher for fast encryption on common processors,” *Inf. Secur. Appl.*, Springer, vol. 8267, pp. 3-27, 2014.

[4] D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B.-S. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, H. Kim, J. Kim, and S. Chee, “HIGHT: A new block cipher suitable for low-resource device,” *CHES 2006*, Springer, pp. 46-59, 2006.

[5] T. Shirai, K. Shibutani, T. Akishita, S. Moriai, and T. Iwata, “The 128-Bit blockcipher CLEFIA (Extended Abstract),” *FSE 2007,* Springer Berlin Heidelberg, pp. 181-195, 2007.

[6] C. H. Lim and T. Korkishko, “mCrypton – A lightweight block cipher for security of low-cost RFID tags and sensors,” *Inf. Secur. Appl.*, Springer Berlin Heidelberg, vol. 3786 LNCS, pp. 243-258, 2006.

[7] S. Moon, M. Kim, and T. Kwon, “Trends on lightweight encryption for IoT communication environments,” *J. KICS*, vol. 33, no. 3, pp. 80-86, Feb. 2016.

[8] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe, “PRESENT: An ultra-lightweight block cipher,” *CHES 2007,* vol. 4727 LNCS, Springer, pp. 450-466, 2007.

[9] Bo Zhu, *An efficient software implementation of the block cipher PRESENT for 8-bit platforms*(2013), 09. 05. 2016, https://github.com/bozhu/PRESENT-C

[10] H. Suh and H. Kim, “Implementation of lightweight encryption algorithm for IoT,” *KIISC*, vol. 25, no. 2, pp. 12-19, Apr. 2015.

박 원 규 (Won-kyu Pim)

2011년 3월~현재 : 동국대학교 컴퓨터공학부 정보통신공학전공 학사과정

<관심분야> 암호학, 소프트웨어공학, 임베디드 시스템

Guillermo Pallarès Cebrián

September 2009~June 2015 : Bachelor’s Degree in Telecommunication Engineering by the Engineering School at Universitat Autónoma de Barcelona

September 2015~February 2017 : Official Master’s Degree in Telecommunication Engineering by the Engineering School at Universitat Autónoma de Barcelona

<관심분야> Software Engineering, Antenna Design, Cryptography

김 성 준 (Sung-joon Kim)

2011년 3월~현재 : 동국대학교 컴퓨터공학부 정보통신공학 전공 학사과정

<관심분야> 암호학, 소프트웨어공학, 임베디드 시스템

이 강 현 (Kang-hyun Lee)

1992년 2월 : KAIST 전기및전자공학과 학사

1995년 2월 : KAIST 정보및통신공학과 석사

1993년 3월~1996년 10월 : 도남시스템 의료기사업부 대리

1997년 7월~2003년 4월 : 아이솔테크놀로지 연구소 책임

2003년 6월~2004년 6월 : 메디슨 연구소 제품기획팀장

2006년 6월~2008년 4월 : 메디퓨처 연구소장

2008년 8월~2014년 12월 : 루맥스 대표이사

2015년 1월~현재 : 더블에이치 연구소장

<관심분야> 라이트테라피(광치료), 웨어러블, 영상진단

임 대 운 (Dae-woon Lim)

1994년 8월 : KAIST 전기및전자공학과 학사

1997년 2월 : KAIST 전기및전자공학과 석사

2002년 8월 : 서울대학교 전기컴퓨터공학부 박사

1995년 9월~2002년 8월 : LS산전 중앙연구소 선임연구원

2006년 9월~현재 : 동국대학교 정보통신공학과 부교수

<관심분야> 암호학, 제어시스템보안

유 기 순 (Ki-soon Yu)

2007년 2월 : 안동대학교 컴퓨터공학과 학사

2015년 2월 : 동국대학교 정보보호학과 석사

2015년 3월~현재 : 동국대학교 정보통신공학과 박사과정

<관심분야> 암호학, 제어시스템보안,