Dragon 스트림 암호 알고리즘의 하드웨어 구현

김헌욱* · 황기현** · 이훈재**

A FPGA Implementation of Stream Cipher Algorithm Dragon

Hun-Wook Kim* · Hwang-Gi Hyun** · Hoon-Jae Lee**

This research was supported by University IT Research Center Project of Korea, and by the Program for Training of Graduate Students in Regional Innovation.

요 약

Dragon 스트림 암호 알고리즘은 현재 ECRYPT 프로젝트의 일부인 eSTREAM에 참여하여 소프트웨어 분야(Profile 1)의 Phase 1, 2단계를 통과하여 Phase 3단계에 등록된 상태이다. Dragon은 기존의 스트림 암호와 달리 한 개의 워드(32비트)단위의 NLFSR(non-linear feedback shift register)을 사용하고, 128/256 비트의 key와 IV(Initialization Vector)를 입력 받아 64비트의 키 수열을 생성하는 키 수열 발생기(Keystream Generator)이다. 본 논문에서는 Dragon 스트림 암호 알고리즘을 Altera사의 Quartus II 툴을 이용하여 Cyclone III FPGA 소자(EP2C35F672I8)에 구현 및 타이밍 시뮬레이션을 하였고, 그 결과 111 MHz에서 7.1Gbps의 처리량을 보였다.

ABSTRACT

Dragon Stream Cipher is proposed for software base implementation in the eSTREAM project. Now this stream cipher is selected as a phase 3 focus candidate. Dragon is a new stream cipher contructed using a single word based NLFSR(non-linear feed back shift register) and 128/256 key/IV(Initialization Vector). Dragon is the keystream generator that produce 64bits of keystream. In this paper, we present an implementation of Dragon stream cipher algorithm in hardware. Finally, the implementation is on Altera FPGA device, EP3C35F672I and the timing simulation is done on Altera's Quartus II. A result of 111MHz maximum clock rate and 7.1Gbps is throughput is obtained from the implementation.

키워드

DRAGON, Stream Cipher, FPGA, VHDL

I. 서 론

암호 알고리즘은 크게 공개키 암호(public-key cryptosystem)와 대칭키 암호(symmetric cipher)로 구분되어진다. 공개키 암호 알고리즘은 암호화에 사용되어지는 키와 복호화에 사용되어지는 키가 서로 다르고 암호키(공개키),복호키(비밀키)의 전송이 필요 없어 키 관리에 용이하다. 하지만 암호/복호화의 속도가 느려 대용

* 동서대학교 유비쿼터스 IT학과
** 동서대학교 컴퓨터정보공학부

접수일자 : 2007. 4. 24

량의 데이터에는 부적합하여 주로 키 분배, 전자서명, 인증 등에 사용되어진다. 대칭키 암호 알고리즘은 암호화와 복호화에 사용되어지는 키가 동일하고, 속도가 빠르지만, 키의 분배에 있어 안전성이 떨어지는 단점을 가지고 있다.

대칭키 암호는 다시 스트림 암호(stream cipher)와 블록 암호(block cipher)로 분류되어진다. 일반적으로 스트림 암호와 블록 암호의 분류 기준은 암호화가 적용되는 평문의 길이에 따라 분류하였다. 암호화가 되는 평문의 길이가 비트나 문자이면 스트림 암호 방식이고, 두 개 이상일 경우 블록 암호 방식이라고 한다[1]. 스트림 암호 알고리즘은 평문을 키 수열과 Exclusive-OR 연산으로 암호문이 되기 때문에 키 수열을 생성하는 키 수열 생성(keystream generator) 알고리즘이 필요하다.

최근에는 스트림 암호의 고속화 성질과 블록 암호의 안전성을 결합한 워드기반 스트림 암호가 제안되고 있으며, 그 대표적인 암호가 Dragon[3]이다. Dragon 스트림 암호 알고리즘은 ECRYPT 프로젝트의 한 부분인 eSTREAM[2](유럽 표준 스트림 암호 알고리즘 공모)에 참여하여, Phase 1, 2를 통과하고 현재 Phase 3, Profile 1 소프트웨어 분야에서 심사를 받고 있다. 소프트웨어 분야는 고속의 소프트웨어용 암호 알고리즘들이 경쟁을 벌이고 있으며, Profile 2의 하드웨어 분야에서는 제한된 하드웨어에서 사용가능한 스트림 암호 알고리즘들이 등록된 분야로 5000게이트 이하에서 구현 가능한 알고리즘들이 경쟁을 벌이고 있다.

본 논문에서는 eSTREAM Profile 1에 등록된 소프트웨어 분야의 암호 알고리즘 중 Dragon 암호 알고리즘에 대하여 고속 하드웨어 구현 가능성을 검토하고자한다. 즉, FPGA 하드웨어에 구현 가능한지, 필요한 Logic cell의 수, 정상 동작 가능한 최대 클락(clock), 처리량(Throughput)을 검토하고자 한다. 본 논문의 구성은 2장에서 Dragon 암호 알고리즘의 구조에 대해 설명하고, 3장에서는 Dragon 암호 알고리즘의 각 모듈을 VHDL 언어로 구현한다. 마지막으로 4장에서는 하드웨어로 구현된 Dragon 암호 알고리즘을 분석한다.

Ⅱ. Dragon 암호 알고리즘 소개

2.1 구성 및 동작

Dragon은 NLFSR(Non-linear Feedback Shift Register) 기반의 스트림 암호 알고리즘으로 1024 비트의 NLFSR과 업데이트 함수인 F 함수, 64비트 메모리인 M으로 구성되며, 256비트의 Key와 IV(Initialization Vector) 또는 128비트의 Key와 IV를 사용할 수 있다. F 함수는 매 라운드 마다 한 번씩 호출되어 사용되며, 각 라운드 마다 64비트의 랜덤한 의사 난수 키 수열(Pseudo-random Keystream)을 생성한다. 그림 1은 Dragon 스트림 암호 암고리즘을 나타내고 있다.

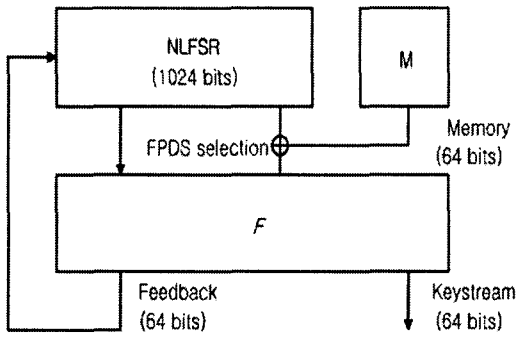

그림 1. Dragon 스트림 암호
Fig. 1. Dragon Stream Cipher

Dragon 스트림 암호 알고리즘은 초기화(Initialization) 상태와 키 수열 생성(Keystream Generation) 상태로 크게 나누어 볼 수 있다. 또한, 그림 1에서 처럼 1024비트 크기의 워드기반 비선형 레지스터(NLFSR), 64비트 메모리 M, 그리고 비선형 F 함수로 나누어진다.

2.2 F 함수

Dragon 스트림 암호 알고리즘의 핵심인 F 함수는 초기화와 키 수열 생성 두 부분에서 동일하게 사용된다. F 함수는 192비트를 입력받아 Pre-mixing, S-Box, Post-mixing 과정을 거쳐 192비트를 출력한다.

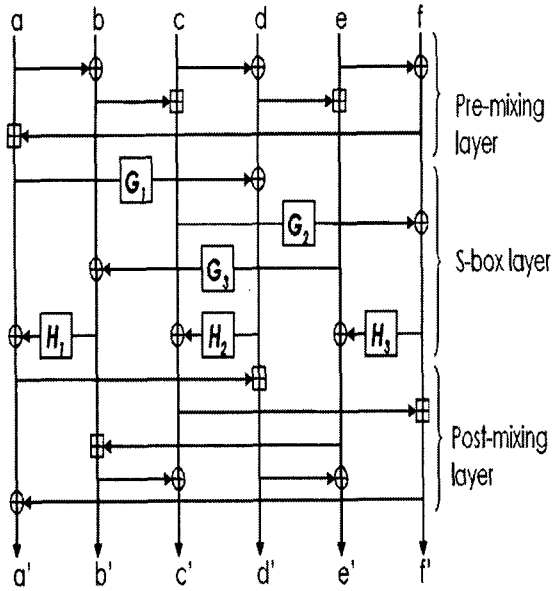

그림 2. F함수
Fig. 2. F Function

F 함수는 그림 2와 같은 순서로 연산되며, 연산 수식은 표 1과 같다. 그림 2와 표 1에서 ⊕는 XOR(exclusive or) 연산이고, ⊞는 mod 2^{32} 덧셈 연산이다.

F 함수의 S-box Layer는 G와 H 함수로 구성되는데, G와 H 함수는 다시 표 2처럼 G_1, G_2, G_3, H_1, H_2, H_3으로 구성된다. G와 H 함수는 비선형(non-linear) 속성을 가진 함수들로 8*32 비트의 S_1,S_2 2개의 S-box들로 구성되어 있으며, 116의 비선형도를 가지고 있다. 미국 국립 표준 기술 연구소(NIST)가 DES의 차세대 국제 표준 암호로 채택한 AES[4]의 S-box는 비선형도가 112로 Dragon의 S-box가 보다 더 높은 값을 가진다. 각 각의 S-box는 256개의 32비트 값을 가지는 1차원 배열로 구성되어 있다. F 함수에 입력되는 6개의 32비트 값은 각 S-box에 입력될 값으로 한 개의 32비트 값(x)은 다시 $x_0 \| x_1 \| x_2 \| x_3$로 구성되어지며 x_0, x_1, x_2, x_3는 각 배열의 인덱스 번호로 사용된다.

표 1. F 함수 연산
Table 1. F Function Operation

Input = {a,b,c,d,e,f}

Pre-mixing Layer

1. b = b ⊕ a;	*d = d ⊕ c;*	*f = f ⊕ e;*
2. c = c ⊞ b;	*e = e ⊞ d;*	*a = a ⊞ f;*

S-box Layer

3. d = d ⊕ G_1(a);	*f = f ⊕ G_2(c);*	*b = b ⊕ G_3(e);*
4. a = a ⊕ H_1(b);	*c = c ⊕ H_2(d);*	*e = e ⊕ H_3(f);*

Post-mixing Layer

5. d' = d ⊞ a;	*f' = f ⊞ c;*	*b' = b ⊞ e;*
6. c' = c ⊕ b;	*e' = e ⊕ d;*	*a' = a ⊕ f;*

Output = {a',b',c',d',e',f'}

표 2. G 와 H 함수
Table 2. G and H Function

$$G_1(x) = S_1(x_0) \oplus S_1(x_1) \oplus S_1(x_2) \oplus S_2(x_3)$$

$$G_2(x) = S_1(x_0) \oplus S_1(x_1) \oplus S_2(x_2) \oplus S_1(x_3)$$

$$G_3(x) = S_1(x_0) \oplus S_2(x_1) \oplus S_1(x_2) \oplus S_1(x_3)$$

$$H_1(x) = S_2(x_0) \oplus S_2(x_1) \oplus S_2(x_2) \oplus S_1(x_3)$$

$$H_2(x) = S_2(x_0) \oplus S_2(x_1) \oplus S_1(x_2) \oplus S_2(x_3)$$

$$H_3(x) = S_2(x_0) \oplus S_1(x_1) \oplus S_2(x_2) \oplus S_2(x_3)$$

2.3 초기화

Dragon 암호 알고리즘은 NLFSR의 내부 초기 값(*W*)이 예측 할 수 없는 값들로 구성하기 위하여 키 갱신(rekeying) 과정을 거치게 되며. 표 3의 초기화 함수를 이용하여 *W* 값을 초기화 한다.

표 3. 초기화 함수
Table 3. Initialization Function

Input = *{ K, IV } (256-bit)*
or
Input = *{ k, iv } (128-bit)*

1. $W_0 \| \ldots \| W_7 = K \| K \oplus IV \| K \oplus IV \| IV$ *(256-bit)*
$W_0 \| \ldots \| W_7 = k \| k' \oplus iv' \| iv \| k \oplus iv' \| k' \| k \oplus iv \| iv' \| k' \oplus iv$ *(128-bit)*
2. $M = 0x0000447261676F6E$

Perform steps 3-8 16 times

3. $a \| b \| c \| d = (W_0 \oplus W_6 \oplus W_7)$
4. $e \| f = M$
5. $\{a', b', c', d', e', f'\} = F(a, b, c, d, e, f)$
6. $W_0 = (a' \| b' \| c' \| d') \oplus W4$
7. $W_i = W_i\text{-}1$, *for* $i = 7$ *down to 1*
(shifting the state by one word)
8. $M = e' \| f'$

Output = *{* $W_0 \| \ldots \| W_7$ *}*

2.4 키 수열 생성

Dragon의 키 수열 생성은 초기화 단계에서 만들어진 W를 32개의 32비트 워드 Bi 로 나누어 표 4 키 수열 함수를 이용하여 최종적으로 64비트 키 수열을 생성한다.

표 4. 키 수열 생성 함수
Table 4. Keystream Generation Function

Input = *{* $B_0 \| \ldots \| B_{31}$, M *}*

1. $(M_L \| M_R) = M$
2. $a = B_0$, $b = B_9$, $c = B_{16}$, $d = B_{19}$, $e = B_{30} \oplus M_L$, $f = B_{31} \oplus M_R$
3. $(a', b', c', d', e', f') = F(a, b, c, d, e, f)$
4. $B_0 = b'$, $B_1 = c'$
5. $B_i = B_{i\text{-}2}$, *31 downto 2*
6. $M = M + 1$
7. $k = a' \| e'$

Output = *{ k,* $B_0 \| \ldots \| B_{31}$, M *}*

III. 하드웨어 구현

Dragon의 하드웨어 설계에 사용된 FPGA 소자는 Altera사의 Cyclone III EP3C25F245C6으로 24,624 개의 Logic Elements를 가지고 있다. 컴파일은 Altera사의 Quatus II 64비트 버전을 사용하였다.

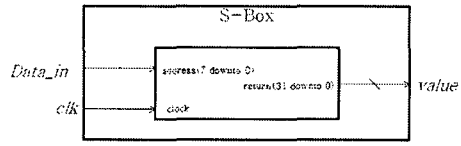

그림 3. S-Box 모듈의 블록
Fig. 3. Block of S-Box Module

```
package dragon_lib is

    function s1(data_in : std_logic_vector(7 downto 0))
            return std_logic_vector;

    function s2(data_in : std_logic_vector(7 downto 0))
            return std_logic_vector;

end dragon_lib;

package body dragon_lib is
    function s1(data_in : std_logic_vector(7 downto 0))
            return std_logic_vector is
            variable val : std_logic_vector(31 downto 0);
    begin
        case data_in is
     when "00000000" => val := x"393BCE6B";
     when "00000001" => val := x"232BA00D";
...
     when "11111110" => val := x"14D79DAC";
     when "11111111" => val := x"0192B555";
        end case;
        return val;
    end s1;

    function s2(data_in : std_logic_vector(7 downto 0))
            return std_logic_vector is
            variable val : std_logic_vector(31 downto 0);
    begin
        case data_in is
     when "00000000" => val := x"A94BC384";
     when "00000001" => val := x"F7A81CAE";
...
     when "11111110" => val := x"96516F9A";
     when "11111111" => val := x"51561E77";
        end case;
        return val;
    end s2;
end dragon_lib;
```

그림 4. S-Box 모듈 코드
Fig. 4. S-Box Module Code

3.1 S-Box 모듈 설계

8비트의 어드레스 값을 입력받아 32비트의 값을 리턴하는 함수로 구현하였으며, 이 함수는 Quartus 툴에서 컴파일시 mif(rom) 파일로 변환된다. 그림 3은 S-box 모듈의 블록 다이어그램이고, 표 5는 VHDL 코드이다.

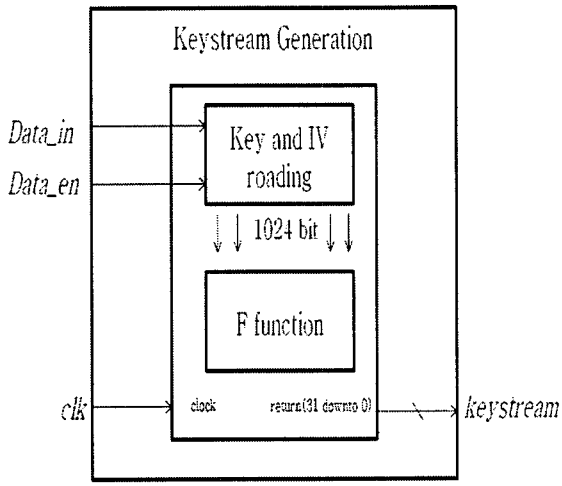

그림 5. 키 수열 생성 모듈의 블록
Fig. 5. Block of keystream generation Module

3.2 키 수열 생성 모듈 설계

키 수열 생성 모듈에서는 초기화 부분을 대신하기 위해 키 로딩 부분을 별도로 설계하였다. 초기화된 wtemp는 표 4의 과정을 거쳐 최종적으로 키 수열을 출력한다. 그림 5는 키 수열 생성 모듈의 블록이며, 그림 6은 키 수열 생성 모듈의 VHDL 코드이다.

그림 6의 1번은 리셋 부분이고, 2번은 초기화 과정을 랜덤한 값을 입력할 수 있도록 하였다. 3번은 F 함수의 pre-mixing 과정이다. 4번은 s-box 연산으로 G와 H 함수가 구현되었다. 5번은 post-mixing 과정이고, 6번의 연산을 거쳐 최종적으로 64비트의 키 수열을 생성하게 된다.

```
library work;
use work.dragon_lib.all;

entity dragon_keygen is
    port(
            clk         :    in std_logic;
            data_in     :    in std_logic;
            data_en :    in std_logic;
            reset   :    in std_logic;
            k           :    out std_logic_vector(63 downto 0)
    );
end dragon_keygen;

ARCHITECTURE ac_dragon_keygen OF dragon_keygen IS
begin
    process(clk, data_in, data_en,reset)
        variable wtemp : std_logic_vector(1023 downto 0);
        variable mtemp : std_logic_vector(63 downto 0) :=x"0000_4472_5167_6F6E";
        variable a,b,c,d,e,f : std_logic_vector(31 downto 0);
    begin
        if reset ='1' THEN                                                    1
            wtemp := x"00000000000000000000 ... 00000000000000000";
            mtemp :=x"0000_4472_5167_6F6E";
        elsif clk'event AND clk='1' THEN
            if data_in = '1' and data_en = '1' then                           2
                wtemp(1023 downto 1):= wtemp(1022 downto 0);
                wtemp := wtemp(1023 downto 1) & data_in;
            else
                a := wtemp(31 downto 0);
                b := wtemp(319 downto 288);
                c := wtemp(543 downto 512);
                d := wtemp(639 downto 608);
                e := mtemp(31 downto 0) xor wtemp(991 downto 960);
                f := mtemp(63 downto 32) xor wtemp(1023 downto 992);

                -- pre-mixing                                                 3
                b := b xor a;
                d := d xor c;
                f := f xor e;
                c := c + b;
                e := e + d;
                a := a + f;

                --s-box                                                      4
                d := d xor (s1(a(7 downto 0)) xor s1(a(15 downto 8))
                    xor s1(a(23 downto 16)) xor s2(a(31 downto 24)));
                f := f xor (s1(c(7 downto 0)) xor s1(c(15 downto 8))
                    xor s2(c(23 downto 16)) xor s2(c(31 downto 24)));
                b := b xor (s1(a(7 downto 0)) xor s2(a(15 downto 8))
                    xor s1(a(23 downto 16)) xor s1(a(31 downto 24)));
                a := a xor (s2(b(7 downto 0)) xor s2(b(15 downto 8))
                    xor s2(b(23 downto 16)) xor s1(b(31 downto 24)));
                c := c xor (s2(d(7 downto 0)) xor s2(d(15 downto 8))
                    xor s1(d(23 downto 16)) xor s2(d(31 downto 24)));
                e := e xor (s2(f(7 downto 0)) xor s1(f(15 downto 8))
                    xor s2(f(23 downto 16)) xor s2(f(31 downto 24)));

                --post-mixing                                               5
                d := d + a;
                f := f + c;
                b := b + e;
                c := c + b;
                e := e + d;
                a := a + f;

                wtemp(1023 downto 64) := wtemp(959 downto 0);
                wtemp(63 downto 0) := b & c;                                  6
                mtemp := mtemp + 1;

                k <= a & e;
            END IF;
        end if;
    END PROCESS;
END AC_dragon_keygen;
```

그림 6. 키 수열 생성 모듈 코드
Fig. 6. keystream generation Module Code

Ⅳ. 구현 결과

Dragon을 Altera사의 Quartus II 툴을 이용하여 VHDL로 코딩하여 컴파일하였다. 표 5는 컴파일 결과이고, 그림 7은 Dragon에 대한 타이밍 시뮬레이션 파형이다. 사용된 시스템 클럭(clk)는 111 MHz 이며, 이 회로에서 사용된 최대 지연시간은 8.8ns로 조사되었고, 111 MHz에서 최대 속도를 보였다.

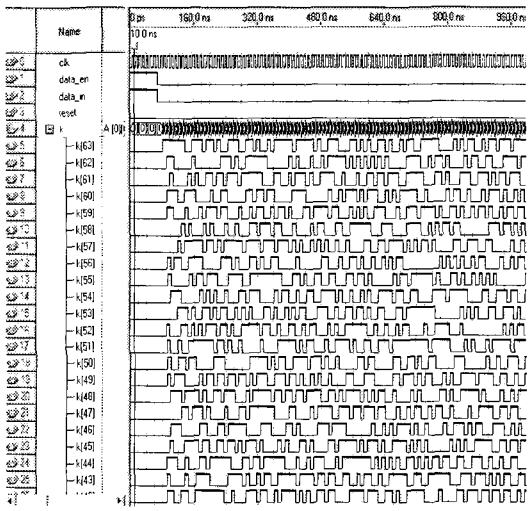

그림 7. Dragon 시뮬레이션 결과
Fig. 7. Dragon Simulation Result

표 5. F 함수 연산
Table. 5 F Function Operation

FPGA Type	MHz	Throughput	Logic Utilization
Cyclon III EP3C25F256C6	111	7.1Gbit/s	Total LE : 21,862 4 LUT :21126 3 LUT : 352 2 LUT : 384

Dragon의 속도는 111MHz 의 주파수에서 동작할 때 7.1Gbps 의 처리량(Throughput)을 보였고, 한 클럭 주기 동안 안전한 데이터를 받기 위해서 92.6MHz의 주파수에서 동작시킬 경우 20% 이상의 시간 여유(margin)를 갖고, 5.9Gbps의 처리량을 보였다. 이 결과는 eSTREAM Profile 2 하드웨어 분야에 등록된 스트림 암호 알고리즘과 속도 측면에서 우월하다고 할 수 있다. 표 6은 eSTREAM Phase 3 Profile 2에 등록된 스트림 암호 알고리즘과 AES-128과 Dragon의 사용 클럭과 속도 비교표이다[5,6,7,8].

표 6. F 함수 연산
Table. 6 F Function Operation

	Clock (MHz)	처리량(Throughput) (Mb/s)
Dragon	111	7100
Edon80	286	3.58
Grain-128	181	181
LEX	100	1454
MICKEY-128	200	200
Phelix	62.5	1000
Salsa20-dr	30	1280
Trivium-1	295	295
Trivium-64	255	15680
AES-128	105	611

즉, Dragon 스트림 암호 알고리즘은 소프트웨어에 적합한 스트림 암호 알고리즘이긴 하지만, 하드웨어로 구현하였을 때 작동 클럭에 비해 빠른 처리율을 보임을 알 수 있다.

Ⅴ. 결 론

Dragon 스트림 암호는 기존의 스트림 암호와 달리 1 클럭에 64비트의 키 수열을 생성하여 64비트로 평문을 암호화 할 수 있다. 본 논문에서는 Dragon을 Altera Quartus II 툴을 이용하여 FPGA 하드웨어(Cyclone III :EP3C25F256C6)에 구현하여 리소스와 속도를 분석하였다. Dragon이 21,862 LE 의 비교적 많은 리소스를 사용하지만 111MHz 클럭에서 7.1Gbps 의 데이터 처리율을 보여준다. 현재 eSTREAM Phase 3의 Profile 2(하드웨어)분야에 응모된 8개의 암호알고리즘은 적은 리소스를 사용하지만 비교적 빠른 클럭을 요구했고, 빠른 클럭에 비해 처리율은 클럭 속도와 비슷하거나 약간 빠름을 보여줬다.

향후, Dragon을 병렬 구현 및 최적화 방안을 마련해서 좀 더 적은 리소스에서 더 빠른 속도를 낼 수 있도록 하는 연구를 진행하고, 다른 암호 알고리즘을 하드웨어에 구현하여 비교 분석할 예정이다.

감사의 글

This research was supported by University IT Research Center Project of Korea, and by the Program for Training of Graduate Students in Regional Innovation.

참고문헌

[1] William Stalings, *"Network and Internetwork Security,"* Principls and Practices, 4 Edition

[2] The eSTREAM web site. *"eSTREAM, ECRYPT Stream Cipher Project,"* http://www .ecrypt.eu.org/st ream.

[3] K.Chen, M.Henrickesen, W.Millan, J.Fuller, L.Simpson, E.Dawon, H.Lee, and S.Moon, *"Dragon : A Fast Word Based Stream Cipher,"* eSTREAM, ECRYPT Stream Cipher Project, http://www.ecrypt.eu.org/stream

[4] Federal Information Processing Standards (FIPS) , *"Advanced Encryption Standard (AES), "* NIST, Technical Report 197, November 2001.

[5] T.Good, W.Chelton and M.Benaissa, *"Review of stream cipher candidates from a low resource hardware perspective,"* eSTREAM, ECRYPT Stream Cipher Project, http://www.ecrypt.eu.org/stream

[6] Philippe Bulens, Kassem Kalach, Franqois-Xavier Standaert and Jean-Jacques Quisquater, *"FPGA Implementations of eSTREAM Phase-2 Focus Candidates with Hardware Profile,"* eSTREAM, ECRYPT Stream Cipher Project, http://www.ecrypt.eu.org/stream

[7] Markus Kasper, Sandeep Kumar and Kerstin Lemke-Rust *"A Compact Implementation of Edon80,"* eSTREAM, ECRYPT Stream Cipher Project, http://www.ecrypt.eu.org/stream

[8] Marcin Rogawski, PROKOM software S.A, *"Hardware evaluation of eSTREAM Candidates: Grain, Lex, Mickey128, Salsa20 and Trivium,"* eSTREAM, ECRYPT Stream Cipher Project, http://www.ecrypt.eu.org/stream

[9] 이훈재, 문상재, *"LILI-II 스트림 암호의 고속화 구현에 관한 연구,"* 한국통신학회논문지 '04-8 Vol.29 No. 8C

저자소개

김 현 욱(Hunwook Kim)

2005년 2월 동서대학교 정보통신 공학 졸업(학사)
2007년 2월 동서대학교 유비쿼터스 네트워크 졸업(석사)
2007년 3월 ~ 현재 : 동서대학교 유비쿼터스 IT 재학
※관심분야 : 네트워크 보안, 암호알고리즘 설계 및 시뮬레이션, 네트워크 설계

황 기 현(Hwang-gi Hyun)

1996년 2월 부산대학교 전기공학과 졸업(석사)
1987년 2월 부산대학교 전기공학과 졸업(박사)
※관심분야 : RFID, 임베디드, 영상처리

이 훈 재(Hoon-jae Lee)

1985년 2월 경북대학교 전자공학과 졸업(학사)
1987년 2월 경북대학교 전자공학과 졸업(석사)
1998년 2월 경북대학교 전자공학과 졸업(박사)
1987년2월 ~ 1998년1월 국방과학연구소 선임연구원
1998년3월 ~ 2002년2월 경운대학교 조교수
2002년3월 ~ 현재 동서대학교컴퓨터정보공학부 부교수
※관심분야 : 암호이론, 네트워크보안, 부채널공격