센서 네트워크 상에서의 HUMMINGBIRD2 암호화 속도 최적화 구현기법

준회원 서 화 정*, 종신회원 김 호 원*°

A Speed Optimized Implementation Technique of HUMMINGBIRD2 Encryption over Sensor Network

Hwa-jeong Seo* *Associate Member*, Ho-won Kim*° *Lifelong Member*

요 약

본 논문에서는 초경량 대칭키 암호화 기법인 HUMMINGBIRD2 알고리즘을 센서 모트상에서의 최적화 구현기법을 제시한다. 효율적인 구현을 위해 센서보드상에 제공되는 레지스터의 활용을 극대화하며 최적화된 주소접근 기법을 적용하여 암복호화에 소요되는 시간을 최소화하였다. 해당 대칭키 암호화 구현기법을 통해 자원 한정적인 센서 상에서의 안전하고 효율적인 보안 통신이 가능하도록 한다.

Key Words : HUMMINGBIRD2, 대칭키 암호기법, 센서 네트워크, MSP430

ABSTRACT

In the paper we present optimized implementation method over sensor mote for HUMMINGBIRD2 algorithm, ultra-light symmetric cryptography. For efficient implementation we maximized the register usage and used optimized addressing method to reduce the encryption and decryption time. With the optimized encryption implementation, we can utilize the efficient secure network over resource constrained sensor mote.

Ⅰ. 서 론

언제 어디서나 통신이 가능한 유비쿼터스 세상의 도래와 함께 이를 실현하기 위한 하나의 기술인 센서 네트워크에 대한 관심이 고조되고 있다. 센서 네트워크는 무선통신이 가능한 센서 모듈들 간에 센싱 된 정보를 교환하면서 정보를 축적해 나가는 형식을 따른다. 특히 센서네트워크는 센싱 및 통신을 하는 장비인 센서 모트(노드) 간에 전송되는 메시지가 무선을 통해 공개된 장소를 통해 전송되는 통신 특성 때문에 도청, 메시지 변·위조와 같은 많은 공격에 노출되어 그 안전도가 위협받고 있을 뿐 아니라 기존의 네트워크 망에서 제기되지 않았던 특수한 문제점을 가진다[1]. 따라서 전송되는 메시지는 대칭키로 암호화하여 메시지를 안전하게 전송해야 한다. 많은 대칭키 암호화 알고리즘 중에서도 최근에 주목받고 있는 알고리즘은 HUMMINGBIRD2이다[2]. 이는 기존의 HUMMINGBIRD1에서 제안되었던 블록 암호화와 스트림 암호화를 조합하는 기법을 통해 초경량 구현이 가능하도록 하여 소프트웨어 및 하드웨어 관점에서 장점을 가지며 동시에 기존의 취약점을 개선하여 보다 안전한 통신이 가능하게 하는 장점을 가진다[3-5]. 따라서 현재 HUMMINGBIRD2는 차세대 무선 USN으로 주목받

※ "본 연구는 지식경제부 및 한국산업기술평가관리원의 산업융합원천기술개발사업(정보통신)의 일환으로 수행하였음. [10039953, 네트워크 중심의 차세대 능동형 RFID 기술 개발]"
* 부산대학교 컴퓨터공학과(hwajeong@pusan.ac.kr, howonkim@pusan.ac.kr) (° : 교신저자)
논문번호 : KICS2011-09-419, 접수일자 : 2011년 9월 29일, 최종논문접수일자 : 2012년 5월 29일

고 있는 Dash7에서 가장 우수한 대칭키 암호로써 고려되고 있다[6]. 본 논문에서는 해당 알고리즘의 구현기법과 과정을 분석한 뒤 소프트웨어 관점에 효과적인 구현을 제시한다. 논문의 구성은 다음과 같다. 2장에서는 HUMMINGBIRD2의 암호화 과정과 구현환경에 대해 알아본다. 3장에서는 현재까지 제시되고 알려진 HUMMINGBIRD2의 구현기법에 대해 알아본다. 4장에서는 제안하는 센서 모트 상에서의 구현기법에 대해 알아본다. 5장에서는 성능을 비교하며 마지막으로 6장에서는 본 논문의 결론을 내린다.

Ⅱ. 논문 인용의 예

본 장에서는 HUMMINGBIRD2 알고리즘에 대한 소개와 암호화 모듈 구현 환경에 대해 설명한다.

2.1. HUMMINGBIRD2

HUMMINGBIRD2는 16-비트 블록 단위로 동작하며 작은 단위의 메시지를 전송할 때 보다 효율적으로 동작한다. 따라서 RFID와 센서네트워크와 같은 환경에 적합하게 설계되었다. HUMMINGBIRD2는 현재 알려진 모든 공격기법에 안전하도록 설계되어 앞으로의 효용가치가 더욱 높다.

2.1.1. 사용되는 용어 및 기호

알고리즘에서 사용되는 용어는 다음과 같다.

표 1. HUMMINGBIRD2 알고리즘 피라메터
Table 1. Parameters in HUMMINGBIRD2 algorithm

용어	설명
K	128-비트 비밀키
R	128-비트 내부 상태
IV	64-비트 초기화 벡터
P_i	16-비트 평문
S	S box
+	2^{16} 모듈러 덧셈기
$\oplus$	exclusive-or 연산기
$\lll$	왼쪽 회전연산(rotation) 명령어
C_i	16-비트 암호문

[표 2]는 키의 결합(mixing)단계에서 사용되는 S-box의 값에 대해 나타내고 있다. S-box는 4-비트 입력 값을 통해 4-비트의 출력 값을 출력하는 방식으로 키의 결합(mixing)이 이루어지도록 한다.

HUMMINGBIRD2에서는 선형변환(linear transform) $L(x)$와 S-box $S(x)$를 조합한 $f(x)$함수를 수행하여 키에 대한 결합(mixing)이 이루어지도록 한다. 해당 연산들은 다음과 같이 수행된다.

$$S(x) = S_1(x_0)|S_2(x_1)|S_3(x_2)|S_4(x_3)$$
$$L(x) = x \oplus (x \lll 6) \oplus (x \lll 10)$$
$$f(x) = L(S(x))$$

정의된 $f(x)$함수는 16비트 키 치환 $WD16(x,a,b,c,d)$에 사용되게 된다. 해당 연산은 다음과 같이 정의된다.

$$WD16(x,a,b,c,d) = f(f(f(f(x \oplus a) \oplus b) \oplus c) \oplus d)$$

2.1.2. 초기화과정

128 비트로 구성되어 상태를 나타내는 R은 초기화 벡터로 초기화된다.

$$R^{(0)} = (IV_1, IV_2, IV_3, IV_4, IV_1, IV_2, IV_3, IV_4)$$

초기화 작업이 수행한 뒤에는 i의 값을 0부터 3까지 증가시키며 다음을 수행한다.

$$t_1 = WD16(R_1^{(i)} + (i), K_1, K_2, K_3, K_4)$$
$$t_2 = WD16(R_2^{(i)} + t_1, K_5, K_6, K_7, K_8)$$
$$t_1 = WD16(R_3^{(i)} + t_2, K_1, K_2, K_3, K_4)$$
$$t_1 = WD16(R_4^{(i)} + t_3, K_5, K_6, K_7, K_8)$$
$$R_1^{(i+1)} = (R_1^{(i)} + t_4) \lll 3$$
$$R_2^{(i+1)} = (R_2^{(i)} + t_1) \ggg 1$$
$$R_3^{(i+1)} = (R_3^{(i)} + t_2) \lll 8$$
$$R_4^{(i+1)} = (R_4^{(i)} + t_3) \lll 1$$
$$R_5^{(i+1)} = R_5^{(i)} \oplus R_1^{(i+1)}$$
$$R_6^{(i+1)} = R_6^{(i)} \oplus R_2^{(i+1)}$$
$$R_7^{(i+1)} = R_7^{(i)} \oplus R_3^{(i+1)}$$
$$R_8^{(i+1)} = R_8^{(i)} \oplus R_4^{(i+1)}$$

2.1.3. 암호화과정

평문 P_i를 암호문 C_i로 생성하기 위해서는 다음과 같이 $WD16$연산을 4번 수행해야 한다.

$$t_1 = WD16(R_1^{(i)} + P_i, K_1, K_2, K_3, K_4)$$
$$t_2 = WD_{16}(R_2^{(i)} + t_1, K_5 \oplus R_5^{(i)}, K_6 \oplus R_6^{(i)}, K_7 \oplus R_7^{(i)}, K_8 \oplus R_8^{(i)}$$
$$t_3 = WD16(R_3^{(i)} + t_2, K_1 \oplus R_5^{(i)}, K_2 \oplus R_6^{(i)}, K_3 \oplus R_7^{(i)}, K_4 \oplus R_8^{(i)}$$

표 2. HUMMINGBIRD2에 사용된 S-box
Table 2. S-box used in HUMMINGBIRD2

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S_1(x)$	7	12	14	9	2	1	5	15	11	6	13	0	4	8	10	3
$S_2(x)$	4	10	1	6	8	15	7	12	3	0	14	13	5	9	11	2
$S_3(x)$	2	15	12	1	5	6	10	13	14	8	3	4	0	11	9	7
$S_4(x)$	15	4	5	8	9	7	2	1	10	3	0	14	6	12	13	11

$$C_i = WD16(R_4^{(i)} + t_3, K_5, K_6, K_7, K_8) + R_1^{(i)}$$

암호문을 생성하고 난 이후에는 다음 과정을 통해 상태 정보를 업데이트 시킨다.

$$R_1^{(i+1)} = R_1^{(i)} + t_3$$
$$R_2^{(i+1)} = R_2^{(i)} + t_1$$
$$R_3^{(i+1)} = R_3^{(i)} + t_2$$
$$R_4^{(i+1)} = R_4^{(i)} + R_1^{(i)} + t_3 + t_1$$
$$R_5^{(i+1)} = R_5^{(i)} \oplus (R_1^{(i)} + t_3)$$
$$R_6^{(i+1)} = R_6^{(i)} \oplus (R_2^{(i)} + t_1)$$
$$R_7^{(i+1)} = R_7^{(i)} \oplus (R_3^{(i)} + t_2)$$
$$R_8^{(i+1)} = R_8^{(i)} \oplus (R_4^{(i)} + R_1^{(i)} + t_3 + t_1)$$

상태정보는 한 번의 암호문이 생성될 때마다 업데이트가 수행된다.

2.2. 구현 환경

다음은 허밍버드 알고리즘을 구현하기 위해 사용한 센서노드의 구조 및 특성과 성능을 평가하기 위해 사용한 시뮬레이션 환경에 대해 알아본다.

2.2.1. 플랫폼 : MSP430

Texas instrument사에 의해 개발된 Tmote Sky 센서노드는 MSP430 16-비트 프로세서를 장착하여 8.192MHz로 동작한다. 48KB 프로그램 플래시 메모리와 10KB RAM을 가지고 있다. MSP430은 총 12개의 general purpose register를 가지며 27개의 명령어 셋을 가진다[7].

2.2.2. 시뮬레이션 환경

Tmote Sky는 nesC 프로그램 언어를 통해 작성되어 TinyOS 운영체제상에서 동작하게 된다. 성능을 판단하기 위해 사용된 툴은 IAR 임베디드 워크벤치(embedded workbench)이다. MSP430 프로세서 상에서의 어셈블리와 c언어를 시뮬레이션하여 성능을 제시한다.

Ⅲ. 이전 연구결과 및 구현기법

HUMMINGBIRD2를 제안한 Revere Security에서는 소프트웨어의 구현에 다양한 모드(mode)를 두어 개발해 놓았다. 사용자는 용도에 따라 속도와 저장공간을 적절히 가감(trade-off)하여 사용이 가능하다. 해당 모드들을 비교해 보면 16 비트 키 치환이 이루어지는 $WD16$단계에서 차이점이 발생한다. 해당 함수는 선형변환(linear transform)과 S-box로 구성되는데 모드에 따라 해당 연산의 구현 기법이 달라진다. 먼저 S-box의 경우 가장 큰 저장공간을 차지하는 정보로써 이를 어떻게 구현하는가에 따라 구현결과가 달라진다. 즉 S-box는 4-비트씩 4개로 16-비트의 값을 도출하는데 입력값으로 16-비트를 주게 될 경우 저장공간은 늘어나고 속도는 줄어들게 된다. 따라서 이에 대한 적절한 가감기법을 찾아야 한다. 선형변환(linear transform)의 경우 S-box의 구성방법에 따라 미리 수행한 결과값을 저장하여 속도를 높이거나 결과값을 도중에 연산하여 저장공간을 최적화할 수 있다.

3.1. Tiny (저장공간 최소화)[2]

저장공간을 최소화하기 위해 채택한 방법은 16-비트 S-box를 별다른 기법을 적용하지 않고 있는

표 3. Tiny버전의 HUMMINGBIRD2에 사용된 S-box
Table 3. S-box used in Tiny version of HUMMINGBIRD2

```
const HB2_u8 SBOX[4][16] ={
{ 0x7,0xC,0xE,0x9,0x2,0x1,0x5,0xF,0xB,0x6,0xD,0x0,0x4,0x8,0xA,0x3 },
{ 0x4,0xA,0x1,0x6,0x8,0xF,0x7,0xC,0x3,0x0,0xE,0xD,0x5,0x9,0xB,0x2 },
{ 0x2,0xF,0xC,0x1,0x5,0x6,0xA,0xD,0xE,0x8,0x3,0x4,0x0,0xB,0x9,0x7 },
{ 0xF,0x4,0x5,0x8,0x9,0x7,0x2,0x1,0xA,0x3,0x0,0xE,0x6,0xC,0xD,0xB }};
```

표 4. Fast버전의 HUMMINGBIRD2에 사용된 S-box
Table 4. S-box used in Fast version of HUMMINGBIRD2

```
const HB2_u8 SBOX12[256] ={
 0x74,0x7A,0x71,0x76,0x78,0x7F,0x77,0x7C,0x73,0x70,0x7E,0x7D,0x75,0x79,0x7B,0x72,
 0xC4,0xCA,0xC1,0xC6,0xC8,0xCF,0xC7,0xCC,0xC3,0xC0,0xCE,0xCD,0xC5,0xC9,0xCB,0xC2,
 0xE4,0xEA,0xE1,0xE6,0xE8,0xEF,0xE7,0xEC,0xE3,0xE0,0xEE,0xED,0xE5,0xE9,0xEB,0xE2,
 0x94,0x9A,0x91,0x96,0x98,0x9F,0x97,0x9C,0x93,0x90,0x9E,0x9D,0x95,0x99,0x9B,0x92,
 0x24,0x2A,0x21,0x26,0x28,0x2F,0x27,0x2C,0x23,0x20,0x2E,0x2D,0x25,0x29,0x2B,0x22,
 0x14,0x1A,0x11,0x16,0x18,0x1F,0x17,0x1C,0x13,0x10,0x1E,0x1D,0x15,0x19,0x1B,0x12,
 0x54,0x5A,0x51,0x56,0x58,0x5F,0x57,0x5C,0x53,0x50,0x5E,0x5D,0x55,0x59,0x5B,0x52,
 0xF4,0xFA,0xF1,0xF6,0xF8,0xFF,0xF7,0xFC,0xF3,0xF0,0xFE,0xFD,0xF5,0xF9,0xFB,0xF2,
 0xB4,0xBA,0xB1,0xB6,0xB8,0xBF,0xB7,0xBC,0xB3,0xB0,0xBE,0xBD,0xB5,0xB9,0xBB,0xB2,
 0x64,0x6A,0x61,0x66,0x68,0x6F,0x67,0x6C,0x63,0x60,0x6E,0x6D,0x65,0x69,0x6B,0x62,
 0xD4,0xDA,0xD1,0xD6,0xD8,0xDF,0xD7,0xDC,0xD3,0xD0,0xDE,0xDD,0xD5,0xD9,0xDB,0xD2,
 0x04,0x0A,0x01,0x06,0x08,0x0F,0x07,0x0C,0x03,0x00,0x0E,0x0D,0x05,0x09,0x0B,0x02,
 0x44,0x4A,0x41,0x46,0x48,0x4F,0x47,0x4C,0x43,0x40,0x4E,0x4D,0x45,0x49,0x4B,0x42,
 0x84,0x8A,0x81,0x86,0x88,0x8F,0x87,0x8C,0x83,0x80,0x8E,0x8D,0x85,0x89,0x8B,0x82,
 0xA4,0xAA,0xA1,0xA6,0xA8,0xAF,0xA7,0xAC,0xA3,0xA0,0xAE,0xAD,0xA5,0xA9,0xAB,0xA2,
 0x34,0x3A,0x31,0x36,0x38,0x3F,0x37,0x3C,0x33,0x30,0x3E,0x3D,0x35,0x39,0x3B,0x32};
```

그대로 구현하는 것이다. 즉 [표 2]에 나타난 S-BOX를 [표 3]과 같이 소스코드 상에 구현함으로써 저장공간을 줄여서 구현하는 방안이다. 하지만 S-box연산은 4개의 4-비트연산들로 구성되므로 한 번의 연산을 수행할 때마다 4번의 메모리 접근이 필요하게 된다. 따라서 저장공간 면에서는 이득이지만 속도는 느려지게 된다.

3.2. Fast (저장공간과 속도의 trade-off)[2]

두 번째 모드는 저장공간과 속도의 적절한 trade-off를 한 모드이다. 여기서는 이전의 Tiny와는 달리 8-비트의 출력값을 출력해 주는 방식으로 구현되어 있다. 따라서 이전에 16개씩 4개의 S-box가 필요했다면 Fast 모드에서는 256(16X16)개씩 2개의 S-box로 구현이 되게 된다. 따라서 저장공간 면에서 이전 모드에 비해 많은 부분을 차지하게 되지만 4번의 메모리 접근을 2번으로 줄였기 때문에 성능이 약 2배 이상 향상되게 된다. 이에 대한 구현은 [표 4]와 같다.

표 5. Furious버전의 HUMMINGBIRD2에 사용된 S-box
Table 5. S-box used in Furious version of HUMMINGBIRD2

```
const HB2_u16 fwhitab[256] = {
0x75cd,0xfbf6,0x30d8,0xf7c5,0x79fe,0xbee3,0xb6c1,0x7def,
0xb2d0,0x71dc,0xffe7,0x3ceb,0x34c9,0x38fa,0xbaf2,0xf3d4,
0xc721,0x491a,0x8234,0x4529,0xcb12,0x0c0f,0x042d,0xcf03,
0x003c,0xc330,0x4d0b,0x8e07,0x8625,0x8a16,0x081e,0x4138,
0xe7a9,0x6992,0xa2bc,0x65a1,0xeb9a,0x2c87,0x24a5,0xef8b,
0x20b4,0xe3b8,0x6d83,0xae8f,0xa6ad,0xaa9e,0x2896,0x61b0,
0x9675,0x184e,0xd360,0x147d,0x9a46,0x5d5b,0x5579,0x9e57,
0x5168,0x9264,0x1c5f,0xdf53,0xd771,0xdb42,0x594a,0x106c,
0x2499,0xaaa2,0x618c,0xa691,0x28aa,0xefb7,0xe795,0x2cbb,
0xe384,0x2088,0xaeb3,0x6dbf,0x659d,0x69ae,0xeba6,0xa280,
0x1455,0x9a6e,0x5140,0x965d,0x1866,0xdf7b,0xd759,0x1c77,
0xd348,0x1044,0x9e7f,0x5d73,0x5551,0x5962,0xdb6a,0x924c,
0x5545,0xdb7e,0x1050,0xd74d,0x5976,0x9e6b,0x9649,0x5d67,
0x9258,0x5154,0xdf6f,0x1c63,0x1441,0x1872,0x9a7a,0xd35c,
0xf7ed,0x79d6,0xb2f8,0x75e5,0xfbde,0x3cc3,0x34e1,0xffcf,
0x30f0,0xf3fc,0x7dc7,0xbecb,0xb6e9,0xbada,0x38d2,0x71f4,
0xb6fd,0x38c6,0xf3e8,0x34f5,0xbace,0x7dd3,0x75f1,0xbedf,
0x71e0,0xb2ec,0x3cd7,0xffdb,0xf7f9,0xfbca,0x79c2,0x30e4,
0x6589,0xebb2,0x209c,0xe781,0x69ba,0xaea7,0xa685,0x6dab,
0xa294,0x6198,0xefa3,0x2caf,0x248d,0x28be,0xaab6,0xe390,
0xd765,0x595e,0x9270,0x556d,0xdb56,0x1c4b,0x1469,0xdf47,
0x1078,0xd374,0x5d4f,0x9e43,0x9661,0x9a52,0x185a,0x517c,
0x0411,0x8a2a,0x4104,0x8619,0x0822,0xcf3f,0xc71d,0x0c33,
0xc30c,0x0000,0x8e3b,0x4d37,0x4515,0x4926,0xcb2e,0x8208,
0x4501,0xcb3a,0x0014,0xc709,0x4932,0x8e2f,0x860d,0x4d23,
0x821c,0x4110,0xcf2b,0x0c27,0x0405,0x0836,0x8a3e,0xc318,
0x8631,0x080a,0xc324,0x0439,0x8a02,0x4d1f,0x453d,0x8e13,
0x412c,0x8220,0x0c1b,0xcf17,0xc735,0xcb06,0x490e,0x0028,
0xa6b9,0x2882,0xe3ac,0x24b1,0xaa8a,0x6d97,0x65b5,0xae9b,
0x61a4,0xa2a8,0x2c93,0xef9f,0xe7bd,0xeb8e,0x6986,0x20a0,
0x34dd,0xbae6,0x71c8,0xb6d5,0x38ee,0xfff3,0xf7d1,0x3cff,
0xf3c0,0x30cc,0xbef7,0x7dfb,0x75d9,0x79ea,0xfbe2,0xb2c4};
```

표 6. 본 논문에서 사용한 MSP430의 명령어 집합
Table 6. Instructions of MSP430 used in this paper

설명	표기	동작	#clock
Load to X	$movR_s, X(R_d)$	$(X+R_d) \leftarrow R_s$	4
	$mov@R_s, X(R_d)$	$(X+R_d) \leftarrow (R_s)$	5
Load to label	$movX(R_s), \&\text{label}$	$(label) \leftarrow (X+R_s)$	6
	$movR_s, \&\text{label}$	$(label) \leftarrow R_s$	4
	$mov@R_s+, \&\text{label}$	$(label) \leftarrow (R_s)$ $R_s \leftarrow R_s+2$	5
	$mov@R_s, \&\text{label}$	$(label) \leftarrow (R_s)$	5
Load to Register	$movX(R_s), R_d$	$R_d \leftarrow (X+R_s)$	3
Copy Register	$movR_s, R_d$	$R_d \leftarrow R_s$	1
Clear Register	$clrR_d$	$R_d \leftarrow R_d \oplus R_d$	1
Addition of the registers	$addR_s, R_d$	$R_d \leftarrow R_s+R_d$	1
	$addcR_s, R_d$	$R_d \leftarrow R_s+R_d+C$	1
	$adcR_d$	$R_d \leftarrow R_d+C$	1
R_d : *Destination register* R_s : *Source register* $X, label$: *Indirect Address Register* PC: *Program Counter* C: *Carry bit at status register*			

3.3. Furious (속도의 최적화)[2]

세 번째 모드는 속도를 가장 우선시해서 작성된 모드이다. 여기서는 8-비트의 S-box로 구현을 했을 뿐 아니라 선형변환(linear transform)이 미리 취해진 상태로 구현이 되도록 하였다. 즉 *WD*16에서는 S-box를 취한 후 선형변환(linear transform)을 취하게 되는 데 미리 선형변환(linear transform)이 취해지는 값을 예측해서 S-box를 작성하는 것이다. 따라서 8-비트의 S-box의 값을 예상하게 될 경우 선형변환(linear transform)에 의해 값이 회전연산(rotation)이 되어 16-비트의 값으로 저장되게 되는데 여기서는 이 값을 그대로 표로 저장해서 사용하는 방식을 의미한다. Fast방식에 비해 여러 번의 이동연산(shift)이 줄어들었지만 S-box의 크기가 두 배로 증가하게 되었다. 따라서 속도와 크기의 Trade-off가 일어났음을 확인할 수 있다.

Ⅳ. 센서 모트 상에서의 제안 구현기법

[표 6]에는 본 논문에서 사용한 MSP430의 명령어 집합에 대해 설명하고 있다. 해당 표는 명령어의 동작 및 소비되는 clock에 대해 나타내고 있다. 여기서도 레지스터 간의 연산은 적은 clock cycle이 소모되지만 메모리에 대한 접근이 일어나게 될 경우에는 소비되는 clock이 증가함을 알 수 있다. 메모리와 연관된 연산은 인자를 읽거나 쓸 때 사용되며 레지스터는 산술연산의 인자들을 계산하기 위한 용도로 사용된다.

최적화된 구현을 위해 C코드와 같은 고급언어가 아닌 기계어 어셈블리를 이용하여 코드를 작성하였다. 어셈블리언어를 이용하면 MSP430에서 제공하는 12개의 범용 레지스터를 사용자가 직접 조작하는 것이 가능하다. 레지스터는 하드웨어적 구조적 이유로 메모리에 비해 정보에 대한 접근 속도가 2배에서 4배가 빠르다. 따라서 제한된 레지스터를 자주 사용하는 정보를 저장하는 용도로 활용하면 메모리 접근(access)을 register에 대한 접근으로 대치시킬 수 있어 속도를 향상시킬 수 있다. 본 논문에서는 속도에 중점을 둔 HUMMINGBIRD2의 구현기법을 제안하며 이는 Furious모드에서 제안한 S-box의 구조를 따른다. 즉 선행변환에 대한 연산이 미리 수행되어 이에 대한 연산이 필요하지 않음을 알 수 있다. 보다 최적화된 구현을 위해 12개의 범

용 레지스터를 용도에 따라 [표 6]과 같이 정의하여 사용하도록 한다. 레지스터는 메모리의 주소를 나타낼 수 있으며 인자들의 값을 유지하거나 명령어의 입력값과 출력값을 저장할 수 있도록 레지스터를 사용할 수 있다. S-box주소는 미리 256개의 16-비트 결과값을 테이블에 저장한 뒤 주소에 접근하여 값을 가져오는 방식을 취하며 상태주소는 R로써 하나의 레지스터는 현재 상태와 다음 상태를 나타내도록 한다. 키 주소의 경우에는 순차적으로 증가하면서 키를 인가함으로 하나의 주소에 연속적으로 키를 배치하면 메모리 접근이 순차적으로 일어나서 부하가 줄어들게 된다. 중간 결과값을 저장하는 레지스터는 주소값을 계산하거나 덧셈과 exclusive-or와 같은 연산을 수행하면 생성되는 중간 값을 유지하는 용도로 사용된다. 상태정보는 연산결과 생성되는 t값은 이후에 다시 사용되는 값이기 때문에 이를 메모리가 아닌 레지스터에서 유지시킴으로써 메모리에 대한 접근을 최소화하는 장점을 가진다.

표 7. 레지스터 할당
Table 7. Allocation of registers

r4	r5	r6	r7	r8	r9	r10	r11	r12	r13	r14	r15
상태주소	키주소	S-box 주소		상태정보 저장				중간결과값 저장			

[표 8-10]은 알고리즘 구현 시 최적화기법이 적용된 부분이다. 해당 알고리즘의 표기는 MSP430의 어셈블리 표기법을 따른다.

4.1. WD16연산 과정

WD16연산은 HUMMINGBIRD2를 구성하는 주 연산이다. 이는 암호화과정에서는 총 4번 수행되며 초기화과정에는 총 16번 수행된다. [표 8]에 나와 있는 코드는 WD16연산을 한번 수행할 때 해야 하는 레지스터 세팅과 값의 할당을 나타낸다. [Step 1-3]에서는 연산에 사용되는 상태와 키주소를 나타내고 있다. 값의 할당을 효율적으로 수행하기 위해 주소증가 방식을 통해 메모리 주소값을 자동으로 계산되도록 하였다. [Step 4-5]에서는 WD16연산에 명시된 덧셈과 exclusive-or연산을 수행한다. [Step 6-14]에서는 S-box에 접근하기 위한 주소값을 계산하게 된다. 현재 MSP430은 16-비트 프로세서이다. 하지만 S-box의 크기는 256 바이트이므로 16-비트 중 8-비트를 이용해서 주소를 접근해야 한다. [Step 6]에서는 바이트 단위로 값을 할당하고 있으며 [Step 7]에서는 주소값이 8-비트이므로 2의 배수로 접근하게 된다. 따라서 shift연산을 통해 주소값을 2의 배수로 계산해 주게 된다. [Step 8]에서는 레지스터 14의 상위 바이트에 존재하는 값을 swpb연산을 통해 바꾸어 주게 되고 [Step 9]에서 and연산을 통해 하위단의 값만을 주소값의 오프셋으로 사용할 수 있다. [Step 11, 13]에서는 주소의 기본 주소에 계산된 오프셋을 더해주는 식으로 현재의 S-box의 주소를 결정하게 된다. [Step 15]에서는 S-box에서 불러온 값들을 exclusive-or연산을 통해 결과값을 도출하게 된다.

표 8. WD16연산 과정
Table 8. Process of WD16 operation

```
 1:mov  @r4+, r13
 2:mov  #0x1, r14
 3:mov  @r5+, r15
 4:add r13,r14
 5:xor r15,r14
 6:mov.b r14,r13
 7:rla r13
 8:swpb r14
 9:and #0x00FF, r14
10:rla r14
11:add r6,r13
12:mov  @r13, r15
13:add r7,r14
14:mov  @r14, r13
15:xor r13,r15
```

4.2. 킷값 할당

메모리에 접근하여 값을 가져오는 연산은 레지스터에 접근하는 연산에 비해 소모되는 시간이 많이 걸린다. 더불어 메모리 주소값은 기본주소와 오프셋에 의해 결정되므로 오프셋을 결정하기 위해서는 추가적인 비용이 들게 된다. 따라서 오프셋을 자동으로 할당해 주는 방식을 사용하면 메모리 접근 당 1 clock cycle을 줄일 수 있는 장점이 있다. 이를 적용하기 위해서는 접근해야 하는 메모리 주소가 순차적으로 배치되어 있어야 한다. 킷값 할당의 경우 암호화 과정에서 순차적으로 값을 가져오도록 알고리즘이 정의되어 있으므로 순차적으로 메모리에 접근하여 주소를 증가시키는 기법을 가감 없이 적용하는 것이 가능하다. 따라서 [표 9]과 같이 순차적으로 값을 할당하도록 코드를 작성하였다.

표 9. 킷값 할당
Table 9. Allocation of key value

```
1:mov @r13+,r4
2:mov @r13+,r5
3:mov @r13+,r6
4:mov @r13+,r7
```

4.3. 회전연산

회전연산(rotation)은 값들에 대해 1~2 비트의 값을 회전 이동 시킬 때 사용되는 연산이다. 한 번의 회전 연산을 수행하게 되면 1 clock cycle이 소모되게 된다. 따라서 회전 연산이 1~2 비트가 회전하는 경우 회전 연산을 그대로 사용하는 것이 가장 최선의 방법이다. 하지만 4-비트 이상 값의 회전 이동이 발생하는 경우에는 다른 연산을 이용하여 보다 효율적인 연산이 가능하도록 할 수 있다. MSP430에서는 16-비트의 레지스터에서 상 하 8-비트씩을 상호 교환시켜주는 상호교체(swap)연산을 제공한다. 이를 이용하면 8-비트의 회전 연산을 1 clock cycle 만에 수행할 수 있으므로 보다 효율적인 회전 연산이 가능하다.

표 10. 회전 연산
Table 10. Rotation operation

```
1:rla r4
2:rlc r4
3:swpb r4
```

V. 성능 비교

[표 11]에는 대칭키 암호화 구현 시의 성능을 비교하고 있다. 표에서 나타나는 바와 같이 HUMMINGBIRD2는 암호화과정과 키 셋업 과정에서 다른 알고리즘에 비해 빠르게 수행된다. 현재 가장 많이 사용되고 있는 AES기법에 비해서도 정보를 빠른 시간 안에 암호화하는 것이 가능하다. HUMMINGBIRD2알고리즘은 블락 크기가 16-bit이므로 데이터의 양이 적은 RFID나 센서네트워크와 같은 환경에서 탄력성 있게 적용이 가능한 장점을 가진다. 이는 기존의 HUMMINGBIRD에서의 장점과 속도 향상을 동시에 충족시킬 수 있기에 HUMMINGBIRD2의 활용도는 보다 높아지리라 생각된다.

HUMMINGBIRD2 알고리즘은 구현 시 속도와 크기 면에서 trade-off가 가능한 측면을 가진다. 다음 표는 현재 센서 모트 상에서의 다양한 목적에 따른 암호화 모듈의 구현을 도식화한 것이다. 본 논문에서 제안한 구현 기법은 기존에 제안된 가장 빠른 구현인 "Furious" 모드에 비해 암호와 초기화 과정에서 적은 사이클을 통해 구현이 가능하다. 대신에 inline 어셈블리를 통해 구현됨으로써 코드길이(code size)가 길어지게 되었다. 해당 제안은 for문을 풀어서 썼기 때문에 코드의 길이가 4배 길어지게 된다. 만약 branch와 같은 조건문을 써서 for문을 돌리게 된다면 코드길이(code size)는 1/4로 줄어들고 속도는 10~20 클럭 사이클이 암호와 초기화

표 11. MSP430상에서의 대칭키 암호화 구현 결과[8,9]
Table 11. Implementation Result of Symmetric Cryptography over MSP430[8,9]

Algorithm	Key length [bit]	Block size [bit]	Encryption [kbit/s]	Key-setup [ms]
AES	128, 192, 256	128	290.24	0.6
SAFER	64, 128	64	232.81	1.7
Twofish	128, 192, 256	128	212.15	15.0
RC2	64, 128	64	161.12	1.2
Noekeon	128	128	138.78	0.1
DES	64	64	123.03	16.6
RC5	128	64	121.44	3.3
Kasumi	128	64	119.93	0.3
XTEA	128	64	112.14	0.4
CAST5	40, 80, 128	64	106.95	0.8
Skipjack	80	64	92.04	0.0
KSEED	128	128	91.9	0.8
RC6	128, 192, 256	128	87.51	5.8
HUMMINGBIRD	256	16	107.43	0.6
HUMMINGBIRD2	128	16	365.10	0.2

에서 증가하게 된다. 현재 MSP430은 48KB를 ROM으로 제공하고 있기 때문에 암호화과정이 자주 일어나는 환경에서는 속도를 우선시 하여 12KB로 최적화 구현하는 것도 고려해 볼만한 사항이다.

표 12. MSP430상에서의 HUMMINGBIRD2의 구현에 따른 비교
Table 12. Comparison of HUMMINGBIRD2 depending on implementation option over MSP430

Mode	Encr[Cycle]	Init[Cycle]	Rom Size[Byte]
"Tiny"	1520	5984	770
"Fast"	576	2187	2518
"Furious"	359	1361	3648
proposal	312	1220	11852

Ⅵ. 결 론

본 논문에서는 2011년에 제시된 HUMMINGBIRD2 알고리즘에 제안된 구현기법을 분석하고 보다 센서 모트에 속도측면에서 최적화된 구현기법을 제시한다. 구현기법은 크게 어셈블리를 이용하여 불필요한 연산을 줄이는 기법과 메모리에 대한 접근을 최소화하고 레지스터에 대한 접근을 극대화하는 방안, 사용되는 레지스터의 순서를 잘 조합하여 사용되는 레지스터의 수를 최적화하는 기법 그리고 명령어들을 잘 조합하여 동일한 연산을 보다 적은 연산으로 수행하는 기법을 소개한다. 이를 통해 보다 적은 계산량을 통해서도 동일한 암호화과정이 가능하므로 이전 기법들에 비해 효율적이며 암호화과정에 소모되는 전력소모를 줄어들게 된다. 현재 이 논문에서는 속도의 최적화 관점에서 작성되었다. 하지만 센서네트워크와 RFID에서 고려해야 할 또 다른 측면은 바로 요구되는 메모리 크기에 있다. 따라서 앞으로의 연구방향은 필요한 저장공간을 최적화하여 속도와 저장공간 모두에서 효율적인 암호화 기법구현을 제안하는 데 있다.

References

[1] A. Perrig, J. Stankovic, and D. Wagner, "Security in Wireless Sensor Networks," *Communications of the ACM*, vol. 47, no. 6, pp. 53-57, June 2004.

[2] D. Engels, M.-J. O. Saarinen, and E. M. Smith, "The Hummingbird-2 Lightweight Authenticated Encryption Algorithm," to appear in the proceedings of *The 7th Workshop on RFID Security and Privacy - RFIDSec* 2011, Berlin, Germany: Springer-Verlag, 2011.

[3] D. ENGELS, X. FAN, G. GONG, H. HU AND E. M. SMITH. "Ultra-Lightweight Cryptography for Low-Cost RFID Tags: Hummingbird Algorithm and Protocol." *Centre for Applied Cryptographic Research (CACR) Technical Reports,* CACR-2009-29. http://www.cacr.math.uwaterloo.ca/techreports/2009/cacr2009-29.pdf

[4] M.-J. O. SAARINEN. "Cryptographic Analysis of All 4X4 - Bit S-Boxes." *Submitted for publication. Available as IACR* ePrint 2011/218, 2011.

[5] M.O. Saarinen, "Cryptanalysis of Hummingbird-1," *In proc. of Fast Software Encryption (FSE)* 2011, to appear, 2011. Also available at:http://eprint.iacr.org/2010/612.pdf

[6] Dash7 alliance, http://www.dash7.org/

[7] Texas instruments, "MSP430x11x MIXED SIGNAL MICROCONTROLLERS", 2004.

[8] D. ENGELS, X. FAN, G. GONG, H. HU AND E. M. SMITH. "Hummingbird: Ultra-Lightweight Cryptography for Resource-Constrained Devices." *1st International Workshopon Lightweight Cryptography for Resource-Constrained Devices (WLC''2010)*. Tenerife, CanaryIslands, Spain, January 2010

[9] Hyncica, O.; Kucera, P.; Honzik, P.; Fiedler, P."Performance evaluation of symmetric cryptography in embedded systems,", *Intelligent Data Acquisition and Advanced Computing Systems (IDAACS), 2011 IEEE 6th International Conference on*, Prague, pp. 277-282, Sept. 2011.

서 화 정 (Hwa-jeong Seo) 준회원

2010년 2월 부산대학교 정보컴퓨터공학과(공학사)
2010년 2월~2012년 2월 부산대학교 컴퓨터공학부 석사
2012년 3월~현재 부산대학교 컴퓨터공학부 박사
<관심분야> 정보보안, RFID/USN, 암호 이론, VLSI 설계

김 호 원 (Ho-won Kim) 종신회원

1993년 2월 경북대학교 전자공학과(공학사)
1995년 2월 포항공과대학교 전자전기공학과 (공학석사)
1999년 2월 포항공과대학교 전자전기공학과 (공학박사)
2008년 2월 한국전자통신연구원(ETRI) 정보보호연구단 선임연구원 / 팀장
2008년 3월~현재 부산대학교 정보컴퓨터공학부 교수
<관심분야> 스마트그리드 보안, RFID/USN 정보보호 기술, PKC 암호, VLSI 설계, embedded system 보안