센서네트워크에 적용 가능한 HIGHT 알고리즘의 최적화 구현 기법

서화정* · 김호원**

Optimized implementation of HIGHT algorithm for sensor network

Hwa-jeong Seo* · Ho-won Kim**

이 논문 또는 저서는 2011년 교육과학기술부로부터 지원받아 수행된 연구임
(지역거점연구단육성사업/차세대물류IT기술연구사업단)

요 약

유비쿼터스 세상의 도래와 함께 언제 어디서나 네트워크 망에 접속하여 자신에게 필요한 서비스를 이용하는 것이 가능해 졌다. 이는 지역의 센싱 정보와 데이터를 제공하는 센서 네트워크의 발달로 인해 가속화되어 가고 있다. 현재 센서 네트워크는 환경 모니터링, 헬스케어 그리고 홈자동화와 같은 우리 삶의 편의에 큰 기여를 하고 있다. 하지만 기존의 네트워크와는 달리 한정적인 자원을 가진 센서를 통한 무선 통신을 수행함으로써 공격자에게 쉽게 노출되는 단점을 가진다. 따라서 센서 네트워크 상에서의 안전한 보안통신을 위해 통신간에 유통되는 메시지는 대칭키로 암호화되어 전송된다. 지금까지 많은 대칭키 암호화 알고리즘이 연구되어 왔으며 그 중에서도 HIGHT 알고리즘은 하드웨어와 소프트웨어 구현에서 기존의 AES보다 속도측면에서 효율적이다. 따라서 RFID 태그와 센서 노드 그리고 스마트 카드와 같은 자원 한정적인 장비에 적합하다. 본 논문에서는 초경량 대칭키 암호화 알고리즘인 HIGHT 알고리즘의 소프트웨어 최적화 구현 기법을 제시한다.

ABSTRACT

As emergence of the ubiquitous society, it is possible to access the network for services needed to us in anytime and anywhere. The phenomena has been accelerated by revitalization of the sensor network offering the sensing information and data. Currently, sensor network contributes the convenience for various services such as environment monitoring, health care and home automation. However, sensor network has a weak point compared to traditional network, which is easily exposed to attacker. For this reason, messages communicated over the sensor network, are encrypted with symmetric key and transmitted. A number of symmetric cryptography algorithms have been researched. Among of them HIGHT algorithm in hardware and software implementation are more efficient than tradition AES in terms of speed and chip size. Therefore, it is suitable to resource constrained devices including RFID tag, Sensor node and Smart card. In the paper, we present the optimized software implementation on the ultra-light symmetric cryptography algorithm, HIGHT.

키워드

센서네트워크, MSP430, HIGHT, 대칭키암호화

Key word

Sensor network, MSP430, HIGHT, Symmetric cryptography

* 준회원 : 부산대학교 컴퓨터공학과 석사과정(hwajeong@pusan.ac.kr)
** 종신회원 : 부산대학교 컴퓨터공학과 교수

접수일자 : 2011. 05. 08
심사완료일자 : 2011. 06. 15

Ⅰ. 서 론

센서네트워크는 언제 어디서나 네트워크가 가능한 유비쿼터스 환경을 실현하는 핵심 기술로써 최근들어 그 관심이 높아지고 있다. 이는 현대인의 생활 속에 깊이 파고들어 편의성 제공에 필요한 여러 가지 정보를 종합 및 추출하여 다양한 서비스, 헬스케어, 홈 자동화, 환경·군사모니터링등과 같은 다양한 분야에 활발히 활용되고 있다. 현재 센서 네트워크 환경은 IEEE 802.15.4 기반으로 한 ZigBee와 같이 실질적인 구현사례가 제시되고 있어 센서네트워크의 활용도는 점차 확대되고 있다.

하지만 센서네트워크는 노드 간에 전송되는 메시지가 무선으로 전송되는 통신 특성 때문에 도청, 메시지 변·위조와 같은 공격에 노출되어 그 안전도가 위협받고 있다. 또한 센서네트워크는 기존의 망과는 달리 한정적인 자원을 가진 노드들로 네트워크가 구성되어 노드상에 구현가능한 암호화 기법에 대한 제한사항이 많다[1]. 유비쿼터스 환경의 실현을 위해서는 유통되는 정보에 대한 안전성 확보를 위한 보안이 무엇보다 우선시되어야 한다. 이를 방지하기 위해 현 센서 네트워크 상에서는 메시지 전송 시 대칭키로 암호화된 암호문을 전송하도록 권장하고 있다. 현재 ZigBee와 같은 무선 네트워크 표준에서는 대표적인 대칭키 알고리즘인 AES를 메시지 전송 시 메시지 암호화에 사용하도록 권장하고 있다. 따라서 지금 사용되는 대부분의 센서노드들은 노드의 칩셋 단에서 하드웨어 모듈로써 암호화를 제공하고 있다[2].

지금까지 많은 연구들이 센서네트워크 상에서 보다 효율적인 대칭키 구현에 대해 연구하여 이에 대한 해결책을 제시하여 왔다. 그 중에서도 최근에 제시된 HIGHT 암호화 알고리즘은 기존의 AES에 비해 하드웨어 및 소프트웨어 구현 시 속도 관점에서 보다 우수한 성능을 나타낸다. 이에 본 논문에서는 센서노드에서의 구현에 가장 적합한 대칭키 인 초경량 알고리즘 HIGHT를 최적화 소프트웨어 구현해 봄으로써 앞으로의 대칭키 암호화의 센서네트워크 구현 시 보다 효율적인 활용이 가능하도록 한다.

본 논문의 구성은 다음과 같다. 2장에서 HIGHT 알고리즘에 대해 소개하며 구현하게 될 환경에 대하여 설명한다. 3장에서 제안하는 최적화 구현 기법을 제시한다. 4장에서는 구현기법의 성능을 분석 및 제시하며 마지막으로 5장에서 본논문의 결론을 내린다.

Ⅱ. HIGHT 암호화 알고리즘

HIGHT 암호화 기법은 64-bit block 길이의 메시지를 128-bit 키길이로 암호화 하는 기법이다. 이는 유비쿼터스 환경에 이용되는 기술인 radio frequency identification(RFID), ubiquitous sensor network(USN)와 같이 자원 한정적인 장비상에서의 운용 시 하드웨어 및 소프트웨어로의 초경량 구현이 현재의 AES보다 성능이 뛰어나다는 장점으로 인해 그 활용도가 높다. [표 1]은 HIGHT 암호화에 사용되는 용어의 설명이다.

표 1. 용어 설명
Table. 1 Description of the notation

$\boxplus$: 2^8상에서의 덧셈
$\boxminus$: 2^8상에서의 뺄셈
$\oplus$: XOR (eXclusive OR)
$A^{\ll s}$: 8-bit 상에서의 왼쪽으로 s-bit 회전
MK : 마스터 키
SK : 서브 키
WK : Whitening 키
X : Transformation에서 사용되는 중간 인수들
P : 평문
C : 암호문

[표 2]에 나타난 HIGHT 암호화 기법은 키를 생성하는 Initialization과정 (Key Schedule), 평문을 암호화하는 Encryption과정 (Initial Transformation, Round Fuction, Final Transformation), 그리고 암호화의 반대 과정을 수행하여 암호문을 복호화하는 Decryption과정과 같이 3개의 과정으로 구성된다[3].

표 2. HIGHT Encryption
Table. 2 HIGHT Encryption

```
HightEncryption(P, MK) {
  KeySchedule(MK, WK, SK);
```

```
HightEncryption(P, WK, SK) {

InitialTransformation(P, X0, WK3, WK2, WK1, WK0);
  For i = 0 to 31 {
RoundFunction(Xi, Xi+1, SK4i+3, SK4i+2, SK4i+1, SK4i);
 }
  FinalTransformation(X32, C, WK7, WK6, WK5, WK4);
 }
}
```

2.1. Key Schedule

[표 3]에 제시된 키 스캐쥴링 과정은 암호화에 사용되게 될 키에 대한 초기화 과정을 수행하게 된다. 키 스캐쥴링 과정은 크게 Whitening키 생성과 서브키 생성으로 나뉜다.

표 3. Key Schedule
Table. 3 Key Schedule

```
KeySchedule(MK, WK, SK){
 WhiteningKeyGeneration(MK, WK);
 SubkeyGeneration(MK, SK);
}
```

[표 4]의 Whitening과정은 마스터 키의 일정 부분을 whitening 키로 할당하는 과정을 나타낸다.

표 4. Whitening Key Generation
Table. 4 Whitening Key Generation

```
WhiteningKeyGeneration{
 For i = 0 to 7{
  If 0 ≦ i ≦ 3, then WKi ← MKi+12;
  Else, WKi ← MKi-4;
 }
}
```

고정된 값을 128주기로 LFSR을 통해 생성하는 Constant Generation은 [표 5]와 같이 초기에 주어지는 초기값90을 LFSR을 통해 연산된 결과값을 통해 생성하게 된다. 해당 값은 서브키 생성에 사용된다.

표 5. Constant Generation
Table. 5 Constant Generation

```
ConstantGeneration{
 s0←0; s1←1; s2←0; s3←1;
 s4←1; s5←0; s6←1;
 For i = 1 to 127{
  si+6 ← si+2 ⊕ si-1;
  δi ← s6||s5||s4||s3||s2||s1||s0;
 }
}
```

[표 6]의 서브키 생성단계는 마스터키와 이전 단계에서 생성된 고정된 값을 이용하여 암호화에 사용되게 될 서브키를 생성하게 된다. 서브키는 주기 8을 기점으로 마스터키와 고정값을 덧셈연산을 수행한다.

표 6. Subkey Generation
Table. 6 Subkey Generation

```
SubkeyGeneration(MK, SK){
 Run ConstantGeneration
 For i = 0 to 7{
  For j = 0 to 7{
   SK16i+j ← MKj-i mod 8 ⊞ δ16i+j;
  }
  For j = 0 to 7{
   SK16i+j+8 ← MK(j-i mod 8)+8 ⊞ δ16i+j+8;
  }
 }
}
```

2.2. Hight Encryption

암호화 단계에서는 이전 단계에서 생성된 서브키와 whitening 키를 이용하여 평문을 암호화하는 과정을 수행하게 된다. 암호화 과정은 크게 Initial, Round 그리고 Final Transformation으로 구성된다.

2.2.1. Initial Transformation

[표 7]의 초기 변환 단계에서는 Whitening키와 평문을 Xor과 덧셈 연산을 통해 다음 암호화 과정인 Round Function의 입력 값을 생성한다.

표 7. Initial Transformation
Table. 7 Initial Transformation

$$
\begin{aligned}
&InitialTransformation(P, X_0, WK_3, WK_2, WK_1, WK_0)\{\\
&X_{0,0} \leftarrow P_0 \boxplus WK_0; X_{0,1} \leftarrow P_1; X_{0,2} \leftarrow P_2 \oplus WK_1; X_{0,3} \leftarrow P_3;\\
&X_{0,4} \leftarrow P_4 \boxplus WK_2; X_{0,5} \leftarrow P_5; X_{0,6} \leftarrow P_6 \oplus WK_3; X_{0,7} \leftarrow P_7;\\
&\}
\end{aligned}
$$

2.2.2. Round Function

[표 8]의 Round Function에서는 세션키와 내부의 인자값을 Xor, 덧셈 그리고 F함수를 통해 변환한 값을 결과값으로 돌려주는 연산을 32번 반복하게 된다.

표 8. Round Function
Table. 8 Round Function

$$
\begin{aligned}
&RoundFunction(X_i, X_{i+1}, SK_{4i+3}, SK_{4i+2}, SK_{4i+1}, SK_{4i})\{\\
&X_{i+1,1} \leftarrow X_{i,0}; X_{i+1,3} \leftarrow X_{i,2}; X_{i+1,5} \leftarrow X_{i,4}; X_{i+1,7} \leftarrow X_{i,6};\\
&X_{i+1,0} \leftarrow X_{i,7} \oplus (F_0(X_{i,6}) \boxplus SK_{4i+3});\\
&X_{i+1,2} \leftarrow X_{i,1} \boxplus (F_1(X_{i,0}) \oplus SK_{4i});\\
&X_{i+1,4} \leftarrow X_{i,3} \oplus (F_0(X_{i,2}) \boxplus SK_{4i+1});\\
&X_{i+1,6} \leftarrow X_{i,5} \boxplus (F_1(X_{i,4}) \oplus SK_{4i+2});\\
&\}
\end{aligned}
$$

$$F_0(x) = x^{\lll 1} \oplus x^{\lll 2} \oplus x^{\lll 7}$$
$$F_1(x) = x^{\lll 3} \oplus x^{\lll 1} \oplus x^{\lll 6}$$

2.2.3. Final Transformation

[표 9]에 제시된 마지막 변환과정에서는 이전 단계의 출력값과 whitening 키를 Xor와 덧셈연산을 사용하여 암호문을 생성하게 된다.

표 9. Final Transformation
Table. 9 Final Transformation

$$
\begin{aligned}
&FinalTransformation(X_{32}, C, WK_7, WK_6, WK_5, WK_4)\{\\
&C_0 \leftarrow X_{32,1} \boxplus WK_4; C_1 \leftarrow X_{32,2}; C_2 \leftarrow X_{32,3} \oplus WK_5; C_3 <- X_{32,4};\\
&C_4 \leftarrow X_{32,5} \boxplus WK_6; C_5 \leftarrow X_{32,6}; C_6 \leftarrow X_{32,7} \oplus WK_7; C_7 \leftarrow X_{32,0};\\
&\}
\end{aligned}
$$

2.3. 구현 환경

본 논문에서의 구현에 사용될 센서노드는 대표적인 센서모트인 MSP430으로써 8.192Mhz로 동작하며 48KB의 ROM, 10KB의 RAM을 가지며 16-bit processor이다. 사용자에 의해 조작이 가능한 general register의 수는 총 12개로써 register 4번부터 15번까지를 사용자는 사용이 가능하다. 해당 register는 인자의 주소를 통해 값을 읽어오는 주소연산과 연산에 사용되는 입력과 출력을 저장하기 위해 사용된다. MSP430에서는 보다 효율적인 주소 접근이 가능하도록 [표 10]과 같이 4가지 종류의 주소 할당 방법을 제시한다. 본 논문에서는 레지스터의 값을 포인터 형식으로 사용함과 동시에 주소를 자동으로 증가시키는 방법인 Increment방식을 통해 효율적인 연산이 가능하도록 하였다.

표 10. MSP430의 주소 모드
Table. 10 MSP430 addressing modes

모드	설명
Direct	레지스터에 의해 바로 주소로 접근한다.
Indexed	레지스터를 인덱스로 사용하여 접근한다.
Indirect	레지스터를 포인터형식으로 사용하여 접근한다.
Increment	Indirect방법과 동일하나 인자의 주소값이 할당이 후 바로 다음을 가리키게 된다.

Ⅲ. 구현 기법

본 장에서는 센서 노드상에서의 효율적인 구현 방안에 대한 구체적인 예시를 제시한다.

3.1. Inline-assembly로의 구현

소프트웨어구현에서 효율적인 프로그램 작성을 위해서는 C언어와 같은 고급언어를 사용하기 보다는 Assembly 언어와 같은 저수준의 언어를 이용한 프로그램 작성이 최적화 구현에 적합하다. 이러한 Assembly 언어의 구현은 Inline-assembly형식으로 C언어 중간에 삽입이 가능하다. Assembly언어를 사용하기 위해서는 가장 먼저 센서모트의 성능에 대한 분석이 우선시 되어야 한다. MSP430 모트는 12개의 general register를 가지고 있어 사용자가 자신이 원하는 프로그램을 해당 register를 사용하여 가능하도록 한다. 기본적으로 2~3개의 register는 input과 output의 memory address를 가리키는 포인터로서 역할을 한다. 따라서 나머지 9~10개의 register를 사용하여 연산에 사용되는 값들에 대한 입력 및 출력을 나타내게 된다. 또한 인자들의 값을 얻기 위한

메모리 access를 효율적으로 수행하기 위해 주소 접근 방법은 Increment방식을 사용한다. 이는 접근하는 메모리 주소가 연속적인 주소로 배치된 경우 첫 번째 인자에 접근한 이후에는 주소 값을 증가시킨다. 따라서 사용자는 다음에 접근하게 될 주소 값을 계산하는 오버헤드를 줄일 수 있는 장점이 있다.

3.2. 사용되는 인자들의 순서변경

8-bit 크기로 구성된 HIGHT의 메시지들을 16-bit processor 환경에서 보다 효율적으로 조작하기 위해 두 개의 8bit 메시지를 하나의 16-bit에 저장하는 방법이 효율적이다. 또한 암호화 과정에서는 2개의 8bit인자가 동일한 xor이나 addition 연산이 수행되는 경우가 있으므로 16bit메시지를 묶어서 사용하는 것이 보다 효율적이다.

암호화 과정에서 사용되는 메시지의 종류에는 마스터키(MK), 화이트키(WK), 서브키(SK), Transformation value X가 있다. WhiteningKeygeneration단계에서 마스터 키는 화이트키를 생성하는 데 사용되는 키로써 두 개의 키 사이에는 [표 11]와 같은 관계를 가진다. 이는 암호화 과정의 Initial Transformation 단계에서 두 개의 8bit인자가 동일한 Xor와 Addition 연산을 수행하기 때문에 묶어서 수행하게 되면 보다 효율적인 연산이 가능하게 되는 장점을 가진다. 표에 나타난 괄호()는 16-bit 메시지의 범위를 나타내며 안의 숫자는 인자의 순서를 나타낸다. 첫 번째 줄에서 (2,0)와 (6,4)는 마스터 키의 2번째, 0번째의 인자가 화이트키의 6번째, 4번째 인자와 매치됨을 의미한다.

표 11. MK와 WK의 키 연관성
Table. 11 Matching between Master Key and Whitening Key

MK	WK
(2, 0)	(6, 4)
(3, 1)	(7, 5)
(13, 12)	(1, 0)
(15, 14)	(3, 2)

이와 동일한 방식으로 평문, 서브키, Transformation value X의 경우에는 [표7]의 Inital Transformation과 [표8]의 Round Function의 경우를 고려해서 서로 동일한 연산이 사용되는 경우 같은 16bit의 형식으로 묶어준다. 이를 표현하면 [표12]와 같은 형식으로 나타낼 수 있다.

표 12. 16-bit 메시지 구성
Table. 12 Construction of 16-bit message

형식	메시지 구성
평문	$(P_4, P_0), (P_5, P_1), (P_6, P_2), (P_7, P_3)$
서브키	$(SK_{4i+3}, SK_{4i+1}), (SK_{4i+2}, SK_{4i})$
X	$(X_{i,7}, X_{i,3}), (X_{i,6}, X_{i,2}), (X_{i,5}, X_{i,1}), (X_{i,4}, X_{i,0})$

3.3. 사전 Key연산

HIGHT알고리즘에서는 마스터 키로부터 서브키를 생성하기 이전에 [표5]와 같이 하나의 고정된 값을 LFSR방식을 이용하여 생성하는 Constant Generation이 있다. 이 연산은 HIGHT알고리즘의 프로그램이 차지하는 메모리 공간을 줄이기 위해 제시한 하나의 기법이지만 속도를 향상시키기 위해서는 해당 연산을 암호화때 구현하지 않고 사전에 해당 값을 도출해서 테이블형식으로 관리하여 사용한다면 보다 빠르게 암호화가 가능하다. 해당 값은 128byte의 값으로써 [표6]의 Subkey Generation과정에서 마스터키와 덧셈연산을 통해 서브키값을 도출하는데 사용된다. 본 논문에서는 해당 값에 대한 계산 부하를 없애기 위해 사전에 계산된 값을 테이블 형식으로 관리하여 효율적인 연산이 가능하도록 하였다.

3.4. Bit Rotation

Round Function에서 2개의 Inner function(F_0, F_1)은 주어진 값을 rotation연산을 수행한 후 xor연산을 수행하여 결과값을 도출하는 연산이다. Rotation의 방향은 경우에 따라 왼쪽 혹은 오른쪽으로 수행된다. 왼쪽으로 rotation을 수행하는 경우에는 carry 값이 발생 시 해당 carry 값을 결과값에 더해주는 방식으로 구현을 한다. 반면에 오른쪽으로 rotation이 수행되는 경우에는 carry 값을 결과값의 가장 상위 bit에 더해주어야 한다. 하지만 여기서는 carry값을 더해 주는 연산이 가장 하위 bit일 경우에만 가능함으로 구현이 어렵다. 따라서 오른쪽으로 rotation이 발생하는 경우에는 하나의 결과값을 더두고 carry값이 발생하는 경우 그값을 다른 하나의 결과값을 rotation시키며 더해주는 방식으로 구현을 한다. 이에 대한 assembly 코드는 [표 13]과 같다.

표 13. Rotation 방법
Table. 13 Method of Rotation

Left Rotation	Right Rotation
mov.b src,r5 rla.b r5 adc.b r5	mov.b src, r10 mov r10, r5 rra.b r5 rrc.b r10

Ⅳ. 성능 분석

본 논문에서는 제안된 구현 방식을 C언어로 구현시의 성능과 비교한다. 동시에 현존하는 다른 암호화기법과의 성능 비교를 통해 HIGHT 암호화 기법이 다른 기법에 비해 우수함을 증명함으로써 해당 알고리즘이 센서네트워크 상에서의 구현에 보다 효율적임을 증명한다. [표 14]은 C코드 상에서의 HIGHT 암호화 구현시의 성능 비교를 나타낸다. 어셈블리 언어에 비해 최적화가 힘든 C코드의 경우 사용되는 cycle이 어셈블리에 비해 월등히 많다.

표 14. HIGHT 암호화 구현 clock cycle 비교
Table. 14 Comparison of implementation of HIGHT cryptography in terms of clock cycle

암호화과정	C코드[5]	Assembly언어
Init	6360	1108
Encryption	7730	2885
Decryption	7860	2885

[표 15]는 HIGHT 알고리즘을 수행하기 위해 수행한 명령어의 수와 clock cycle에 대해 나타낸다. 여기서 CPI는 해당 연산을 수행하는데 걸리는 clock cycle을 의미한다. 연산은 명령어, 시작지 그리고 목적지로 구성된다. 여기서 add, xor, rlc, rrc, and, swpb는 덧셈, exclusive-or, 왼쪽 rotation, 오른쪽 rotation, And 그리고 swap연산을 각각 나타낸다.

암호화 연산에 사용되는 전력소비량을 계산하기 위해 $W= U\times I\times t$공식을 사용하여 계산하였다. 여기서 U, I 그리고 t는 각각 전압, 전류, 실행시간을 의미한다. 결과는 [표 16]에 나타난 바와 같다.

표 15. 어셈블리 언어의 clock cycle분석
Table. 15. Analysis of the clock cycle in assembly code

명령어	CPI	제안된 알고리즘			
		명령어 수		Cycle	
		Init	Enc	Init	Enc
mov @reg+,reg	2	14	72	36	144
mov x(reg),reg	3	70	5	216	15
mov reg,x(reg)	4	68	4	288	16
add src,dst	1	287	395	292	395
xor src,dst	1	0	481	0	481
rlc dst	1	0	128	0	128
rrc dst	1	0	768	0	768
swpb dst	1	28	224	28	224
and src,dst	1	220	198	220	198
mov src,dst	1	28	516	28	516
Total		715	2791	1108	2885

표 16. HIGHT 암호화 구현 전력소비 비교
Table. 16 Comparison of the implementation of HIGHT cryptography in terms of power consumption

암호화과정	C코드[5]	Assembly언어
Init	20.68293 mJ	0.753897 mJ
Encryption	25.13821 mJ	1.96299 mJ
Decryption	25.56098 mJ	1.96299 mJ

[표 17]에서는 HIGHT 알고리즘과 최근에 연구 중에 있는 여러 대칭키 암호화 알고리즘의 효율성을 비교해 본다. 모든 알고리즘은 16-bit processor상에서의 구현에서의 성능을 나타낸다. HIGHT는 다른 대칭키 알고리즘에 비해 전체적으로 성능이 빠른 편이다. Humming bird 알고리즘과 비교해 보면 암호화를 한번 수행시에는 HIGHT알고리즘이 초기화와 암호화를 수행하는 연산이 적게 사용된다. 하지만 암호화와 복호화 과정만을 두고 볼 때는 Humming bird가 장점을 가진다.

표 17. 암호화 알고리즘에 따른 clock cycle
Table. 17 clock cycle depending on cryptography algorithm

암호화 알고리즘	초기화	암호화	복호화
Hummingbird[6]	4,824	1,220	1,461
PRESENT[7]	-	19,464	33,354
HIGHT	1108	2885	2885

소비되는 clock cycle에 따른 전력 소모를 알아보면 [표 18]과 같다. 위에서 살펴본 바와 같이 HIGHT 알고리

즘은 전체 암호화과정에서 볼때 다른 암호화 알고리즘에 비해 적은 에너지를 소비한다.

표 18. 암호화 알고리즘에 따른 전력소모
Table. 18 power consumption depending on cryptography algorithm

과정	Humming Bird	PRESENT	HIGHT
Init	3.282309 mJ	-	0.753897 mJ
Enc	0.830103 mJ	13.24355 mJ	1.96299 mJ
Dec	0.994082 mJ	22.69447 mJ	1.96299 mJ

V. 결 론

본 논문에서는 초경량 블락 대칭키 기반 암호화인 HIGHT의 16-bit processor상에서의 최적화 소프트웨어 구현기법에 대해 알아보았다. 제시된 방안은 최초로 제시된 16-bit환경에서의 HIGHT암호화 모듈의 어셈블리 구현으로써 의미가 있다. 이는 HIGHT알고리즘의 센서 네트워크 환경의 플랫폼 상에서의 소프트웨어 구현에서 좋은 성능을 보임을 증명하여 하드웨어와 소프트웨어 구현에 모두 적합함을 알 수 있다. 또한 기존의 다른 대칭키 알고리즘에 비해 속도 측면에서 빠른 연산이 가능하다는 장점을 가진다. 앞으로의 연구 방향은 본 논문의 결과인 속도 최적화 구현을 토대로 하여 센서노드에서 또 하나의 중요한 자원인 메모리소비 관점에서 보다 효율적인 암호화 구현을 제시하는데 있다.

참고문헌

[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] Kinney, P. 2003. "ZigBee Technology: Wireless Control that Simply Works." ZigBee Alliance.

[3] Deukjo Hong, Jaechul Sung, Seokhie Hong, Jongin Lim, Sangjin Lee, Bonseok Koo, Changhoon Lee, Donghoon Chang, Jaesang Lee, Kitae Jeong, Hyun Kim, Jongsung Kim, and Seongtaek Chee. HIGHT: A New Block Cipher Suitable for Low-Resource Device. In Louis Goubin and Mitsuru Matsui, editors, CHES, volume 4249 of Lecture Notes in Computer Science, pp. 46-59. Springer, 2006.

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

[5] W. Liu, R. Luo, and H. Yang. Cryptography overhead evaluation and analysis for wireless sensor networks. Communications and Mobile Computing, International Conference on, pp. 496 - 501, 2009.

[6] D. Engels, X. Fan, G. Gong, H. Hu, and E. M. Smith, "Hummingbird:Ultra-Lightweight Cryptography for Resource- Constrained Devices", to appear in the proceedings of The 14th International Conference on Financial Cryptography and Data Security FC 2010, 2010.

[7] A. Poschmann, Lightweight Cryptography Cryptographic Engineering for a Pervasive World", Ph.D. Thesis, Department of Electrical Engineering and Information Sciences, Ruhr-UniversitÄaet Bochum, Bochum, Germany, 2009.

저자소개

서화정(Hwa-jeong Seo)

2004.3～2010.2 : 부산대학교 정보컴퓨터공학과 학사
2010.2～현재 : 부산대학교 컴퓨터공학부 석사
※관심분야 : 정보보안, RFID/USN, 암호 이론, VLSI 설계

김호원(Ho-won Kim)

1989. 3～1993. 2 : 경북대학교 전자공학과 학사
1993. 3～1995. 2 : 포항공과대학교 전자전기공학과 공학석사
1995. 2～1999. 2 : 포항공과대학교 전자전기공학과 공학박사
1998.12～2008. 2 : 한국전자통신연구원(ETRI) 정보보호연구단 선임연구원 / 팀장
2008. 3～현재 : 부산대학교 정보컴퓨터공학부 조교수
※관심분야 : 스마트그리드 보안, RFID/USN 정보보호 기술, PKC 암호, VLSI 설계, embedded system 보안