# Comparative performance and energy consumption analysis of different AES implementations on a Wireless Sensor Network Node

Fan Zhang, Reiner Dojen*, Tom Coffey

Data Communications Security Laboratory,

University of Limerick,

Limerick, Ireland

Fax: +353-61-338176

Email: fan.zhang@ul.ie

Email: reiner.dojen@ul.ie

Email: tom.coffey@ul.ie

*Corresponding author

**Abstract** – Many of the diverse Wireless Sensor Network (WSN) applications, such as home automation, traffic control and medical patient monitoring, have a strong requirement for the confidentiality of sensing data. At the same time, the restricted resources of WSN nodes demand that applications are implemented as efficiently as possible. Recently, the Advanced Encryption Standard (AES) has been proposed as the preferable symmetric cipher for WSN applications.

This paper presents a performance and energy consumption analysis of three AES implementations on a wireless sensor node. For each implementation the memory requirements, the execution time and the energy consumption is monitored. The results of the presented analysis show that while the hardware supported AES is faster than the software implementations it is less energy efficient, due to increased power requirements of the additional hardware.

**Keywords**: Wireless Sensor Networks, Energy Efficency, Cryptography, MicaZ Sensor Node, Advanced Encryption Standard

**Acknowledgements:** This work funded by the Irish Research Council for Science, Engineering and Technology (IRCSET Embark Initiative) and Science Foundation Ireland - Research Frontiers Programme (SFI RFP07CMSF 631).

**Biographical Notes**: Fan Zhang is currently persuing his PhD in the Department of Electronic and Computer Engineering at the University of Limerick in Ireland. He received title of Bachelor of Science from University of Xiangtan, China, in 2003 and Master of Engineering from University of Limerick, Ireland, in 2007.

Reiner Dojen received title of Diplom-Ingenieur (FH) from the University of Applied Sciences Osnabrück, Germany, in 1999, Master of Engineering from University of Limerick, Ireland, in 2000 and PhD from University of Limerick, Ireland, in 2004. He is currently employed as Lecturer at the Department of Electronic and Computer Engineering at the University of Limerick in Ireland. His past employment includes Software Design Engineer at S.u.S.E. Linux in Nürnberg, Germany, Software Design Engineer at University of Applied Sciences Osnabrück, Germany and Assistant Teacher at University of Limerick.

Tom Coffey holds a Masters Degree from City University London, UK and a Doctor of Philosophy Degree from University of Ulster, Ireland. He is Professor of Electronic and Computer Engineering at the University of Limerick, Ireland and Head of Department. He is founder and director of the Data Communications Security Laboratory at the University of Limerick. He is a Chartered Engineer and his previous employments include: Telecommunications Design Engineer and Manager at Standard Telephones and Cables, London, UK and Research and Development Manager at Telectron Ltd., Dublin, Ireland.

## 1. Introduction

Wireless Sensor Networks (WSN) are ad hoc networks composed of small network nodes with low power capacities, small resource storages and relatively low computation capabilities (Romer and Mattern, 2004). These WSNs have been adopted in many areas, such as home automation, traffic control, assembly line monitoring and medical patient monitoring. These diverse WSN applications impose a wide range of constraints onto the WSN systems, including their physical size, cost, power availability and performance (Chour and Park, 2005). Further, many of these application areas have a strong requirement for the confidentiality of sensing data. As this data is transmitted over wireless connections, securing the communication between wireless network nodes is paramount. At the same time, the limited computational and power resources available on WSN nodes demand that applications are implemented as efficiently as possible. While it can be seen that the power consumption of a WSN application relies largely on the used hardware, it is also affected by the efficiency of the employed software. Thus, careful selection of energy efficient algorithms and protocols can significantly enhance a node’s lifetime.

For many years, RC5 (Rivest, 1995) and Skipjack (NIST, 1998) were considered to be the most suitable cryptographic ciphers for WSN applications (Karlof, Sastry, and Wagner, 2004, Vitaletti and Palombizio, 2007). However, recently the Advanced Encryption Standard (AES) - also called Rijndael (Daemen and Rijmen, 1999) - was proposed as a preferable option (Law, Doumen, and Hartel, 2006, Healy, Newe, and Lewis, 2007), as it provides stronger security protection while still being suitable for the low resource requirements on WSN nodes.

AES can be implemented on WSN nodes either exclusively in software or with hardware support. For the software implementation of AES, this paper considers two algorithms: Firstly, the original Rijndael algorithm (Daemen and Rijmen, 1999) and, secondly, an optimised algorithm that performs several operations of the original algorithm as table lookups. Hardware support for AES is available on many platforms. For example the widely used RF transceiver chip Chipcon CC2420 provides a 128-bit AES encryption functionality (Chipcon, 2004). While an AES implementation with hardware support is faster than software implementations (Healy, Newe and Lewis, 2007), it is not clear whether a faster implementation necessarily implies a more energy efficient implementation, as the extra hardware requires additional power.

This paper presents a performance and energy consumption analysis of three AES implementations: (1) exclusively software AES using the original Rijndael algorithm, (2) exclusively software AES using an optimised table lookup AES and (3) hardware supported AES using the Chipcon CC2420 RF transceiver chip.

This analysis examines the memory requirements, execution times and energy efficiency of the employed algorithms when implemented on a MicaZ sensor node (CrossBow, 2010) running TinyOS 2.1.0 (Levis et al., 2005). The power consumption is measured using the Agilent 66321D Mobile Communications DC source and 14565B Device Characterization Software (Agilent, 2007). In summary, the results of our analysis show that while the hardware supported AES is faster than the software implementations it requires more energy. As a consequence, using an optimized software AES implementation – rather than a hardware supported implementation - can enhance the lifetime of sensor nodes significantly.

## 2. WSNs and Power Conservation

WSN applications often impose a wide range of constraints onto WSN systems, particularly on their physical size, cost, performance and power availability (Chou and Park, 2005). Provision of a suitable power supply is particularly challenging. On one hand, the decrease of physical node sizes and the increase of nodes in WSN systems makes the replacement of depleted batteries more and more impractical. On the other hand, providing a large capacity battery is often not viable, as it would dominate the overall system size. Consequently, the lifetime of the network relates deeply to the lifetime of the sensor node's battery (Sohraby, Minoli and Znati, 2007) and WSN system design needs to focus on power conservation.

### *2.1 WSN Power Consumption Domains*

A wireless sensor node's power consumption can be divided into three domains: sensing, communication, and data processing (Akyildiz et al., 2002). To extend a WSN system's lifetime, designers need to optimise the system across all three domains.

#### *2.1.1 Sensing Domain*

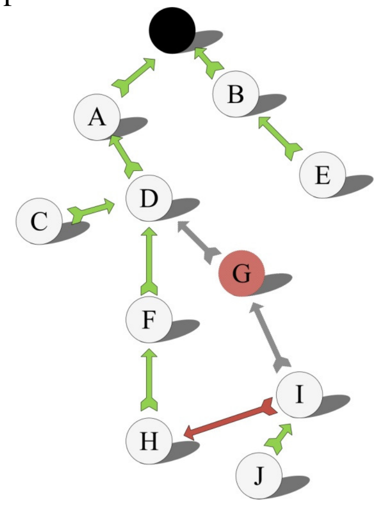
In the sensing domain, the power consumption of a wireless sensor node relies essentially on the used hardware platform. However, depending on the sensing application, factors such as sensing range, sensing frequency and sensing items also influence the power consumption.

#### *2.1.2 Communication Domain*

Radio communication is expected to consume the largest amount of energy (Shih et al., 2001). For example, the MicaZ sensor node with a CC2420 radio transceiver requires more than twice the energy in reception mode than in data processing mode. Hence, radio transceiver actions should be kept as short as possible. Additionally, the radio transceiver's start-up power consumption should not be neglected, as it may consume comparatively large amounts of energy (Akyildiz et al., 2002). However, radio communication power consumption is not only affected by the hardware for radio transmission and reception, but is also influenced by the selection of various protocols and algorithms dealing with items such as routing or clustering.

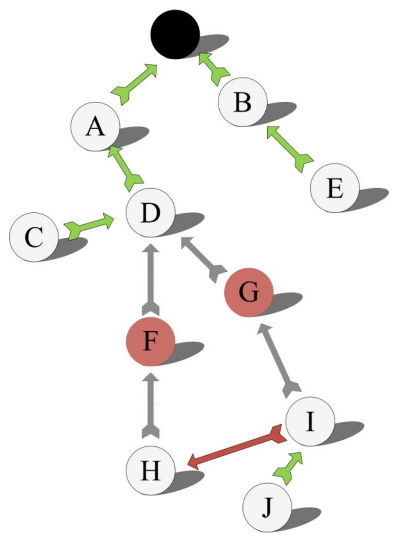
Multi-hop routing is more energy efficient in densely deployed WSN than long distance single hop communication (Yick, Mukherjee, and Ghosal, 2008). Further, multi-hop routing also overcomes signal propagation problems (Akyildiz et al., 2002) and, thus, saves energy spent on the retransmission of failed communications. However, multi-hop routing protocols also have their disadvantages. Most notably, routing nodes that are part of multiple routing paths consume more energy than nodes on only a single path. Consequently, these multi-path nodes have a shorter lifetime, as their batteries drain faster (Sohraby, Minoli and Znati, 2007). A battery drained node (also called a dead node) may cause a significant change in the network topology. For example, if the battery of node "G" in Figure 1 is drained, the nodes "J" and "I" have an increased path to the Base Station, resulting in an overall increased power consumption in the WSN.

**Figure 1:** Dead node causing increased routing path



In a worst case scenario, existence of some dead nodes may cause the isolation of a subset of the WSN. For example, the dead nodes "F" and "G" in Figure 2 cause the isolation of the nodes "H", "I" and "J" (under the assumption that the distance between these nodes and the nearest node of the remaining network "D" is too large for communication).

**Figure 2:** Multiple dead nodes isolating a subnet



*2.1.3 Data Processing Domain*

In the data processing domain, energy is spent on storing and accessing sensing data, control and coordination of the hardware components (including sensing and transmission components and power management) and the preparation of the sensing data for transmission. The latter also includes auxiliary operations for communications, such as compression of data to minimise transmissions, encryption/decryption of data and calculation of hashes. Many techniques have been proposed to reduce the power consumption of a node's processor, i.e. dynamic voltage scaling (Min, Furrer, and Chandrakasan, 2000), dynamic CPU speed setting (Govil, Chan, and Wasserman, 1995) and improving processor time management (Lorch and Smith, 1996).

However, the power consumption for data processing is also heavily affected by the selection of suitable algorithms and their implementation details. Thus, for completing a specific task, such as data encryption, it is important to adopt the most energy efficient solution. Further, programmers need to carefully consider implementation details. For example, local variable usage in WSN programs should be limited, as generating local data tables is a burden on the processor that consumes additional power. Similarity, RAM dumping should be avoided as it is also a time-consuming and energy intensive task (Levis et al., 2005).

In summary, while the power consumption of a WSN node relies largely on the used hardware, it is also influenced by the nodes' software. Thus, node lifetime can be significantly enhanced by using carefully selected algorithms and protocols and energy efficient software implementations (Raghunathan et al., 2002).

## 3. Advanced Encryption Standard (AES)

The U.S. government adopted the Rijndael algorithm (Daemen and Rijmen, 1999) as the new Federal Advanced Encryption Standard (AES), which was published by the National Institute of Standards and Technology (NIST) under U.S. FIPS PUB 197 (FIPS 197) in 2001.

*3.1 AES Algorithm Details*

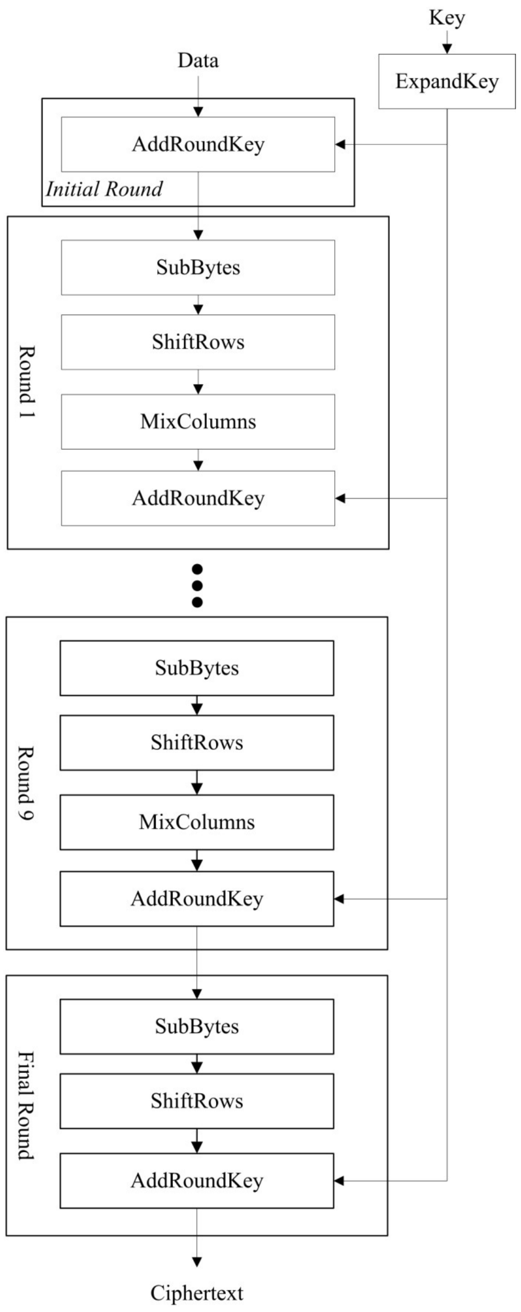
AES's design principle is based on a *substitution permutation network*. AES supports a fixed block size of 128 bits and variable key sizes of 128 bits, 192 bits and 256 bits. The AES cipher initially organises the plaintext into a 4x4 array of bytes (called the *state)* and expands the key into the required number of round keys. Then it performs the initial round followed by a number of repetitions of transformation rounds and a final round (cf. Figure 3). The number of repetitions depends on the key-size: for a 128-bit key 10 rounds are used, for a 192-bit key 12 rounds are used and for a 256-bit key 14 rounds are used. A set of reverse rounds using the same (expanded) encryption key is applied to transform ciphertext back into the original plaintext.

Except for the initial and final round, each round contains the following steps:

- *AddRoundKey*: combining the round key with the state.
- *SubBytes*: Every byte is replaced with another according to Rijndael's *S-Box*
- *ShiftRows*: Shifting each row of the state with a certain offset. The first row remains the same in Rijndael.
- *MixColumns*: Combining the four bytes in each column using an invertible linear transformation.

The initial round performs only the step *AddRoundKey*, while the final round does not perform the *MixColumn* step.

**Figure 3:** AES Encryption Process



*3.2 Security of AES*

AES encryption with a 128-bit key has been verified to be secure enough to protect classified information up to US Government's SECRET level (Hathaway, 2003). While some theoretical weaknesses have been found in AES – e.g. the XSL attack against 128-bit key AES (Courtois and Pieprzyk, 2002) – these require on the order of $2^{100}$ operations and, thus, are regarded as not practical with current technology. 128-bit key AES is considered as sufficient for WSN applications (Healy, Newe, and Lewis, 2007).

*3.3 Optimised AES Implementation*

The performance of the AES algorithm has been widely analysed (Xiao et al, 2006b, Zhang and Niu, 2008). For example, the original Rijndael algorithm contains some redundancy in the shifting or rows and mixing of columns operations, which can be replaced by table lookup operations (Bertoni et al., 2003, Zhang and Niu, 2008). In particular, many AES implementations have been proposed for wireless devices (Xiao et al., 2006a, Diala, Ault, and Bagchi, 2008, Olteanu, Xiao, and Zhang, 2009, Olteanu and Zhang, 2010)

## 4. Performance and Energy Consumption Analysis of AES Implementations

This section presents a comparative performance and energy consumption analysis of three AES implementations to enable WSN application developers to make the most suitable choice. The following AES implementations are used:

1. Exclusively software AES using the original Rijndael algorithm, henceforward called *OriginalAES*.
2. Exclusively software AES using an optimised table lookup AES, henceforward called *TableLookupAES*.
3. Hardware supported AES using the Chipcon CC2420 RF transceiver chip, henceforward called *HardwareAES*.

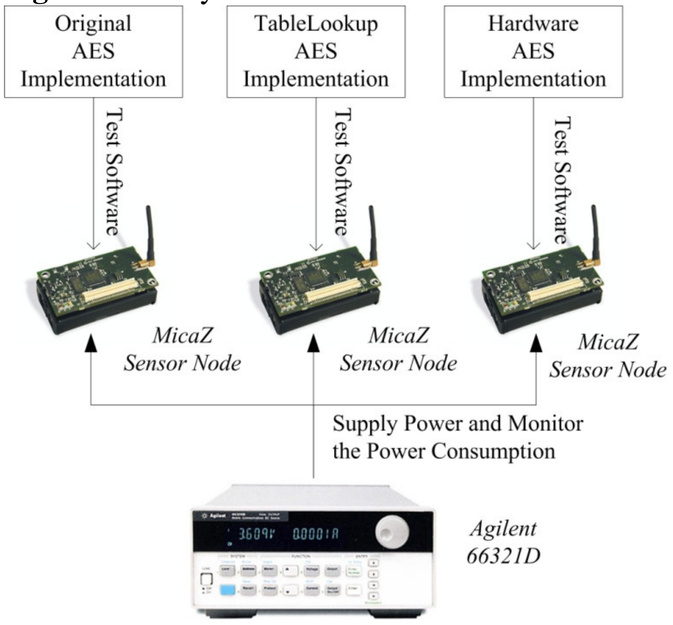
Figure 4 presents an outline of the undertaken analysis, which is divided into three parts:

1. Implementation of AES variants (*OriginalAES, TableLookupAES, HardwareAES).*
2. Development of test application on MicaZ.
3. Performance and power consumption measurements of the test application running on a MicaZ sensor node.

*4.1 Employed System*

The presented analysis is performed using the widely-used MicaZ sensor node (Galindo, Roman, and Lopez, 2008, Kohno, Ohta, and Kakuda, 2009, Xiong, Wong, and Deng, 2009, Borms et al., 2010) as implementation and test platform. The MicaZ sensor node is a 2.4 GHz, IEEE/ZigBee 802.15.4 board used for low-power wireless sensor networks, with 8-bit ATmega128L microcontroller, 128KB RAM, 512KB ROM and a 2.4 GHz Chipcon CC2420 RF transceiver.

**Figure 4:** Analysis outline


Original AES Implementation
TableLookup AES Implementation
Hardware AES Implementation
Test Software
Test Software
Test Software
MicaZ Sensor Node
MicaZ Sensor Node
MicaZ Sensor Node
Supply Power and Monitor the Power Consumption
Agilent 66321D

The Chipcon CC2420 features hardware AES-128 encryption support with multiple modes: standalone mode, counter mode (CTR), CBC-MAC authentication mode and CCM encryption & authentication mode.

Power consumption is monitored using an Agilent 6321D Mobile Communications DC Source and the Agilent 14565B Device Characterization Software. The Agilent 66321D can be used as a DC source to mobile devices such as MicaZ sensor nodes or cell phones. It provides an output voltage of 0-15V and current of 0-3A. While supplying energy to a device, it is able to sample output voltage and current with a time resolution ranging from 15µs to 31,200s. The Agilent 14565B software has multiple functions:

- Controlling the Agilent 66321D to supply power with defined current or voltage.
- Measuring and recording the current and voltage changes at the designated time resolution.
- Generating the current/voltage graph over time.

Combining the Agilent 66321D DC Source with the 14565B Device Characterization Software, the mobile device's energy consumption can be monitored and recorded with high precision (resolution of 2.5µA for current measurements and 3mV for voltage measurement).

The MicaZ nodes run on TinyOS, which is the first operating system designed especially for WSN (Levis et al., 2005). It is an open-source OS that is energy efficient and provides good power management features. For example, it automatically puts the processor into soft-sleep mode (using about 3.5mA for the ATmega128L) when no task is waiting in the queue. For the analysis presented in this paper, TinyOS 2.1.0 is used. The programming language used to develop applications for TinyOS is nesC (Gay et al., 2003), which is an extension of the C language. Further, TinyOS supports a wide range of different sensor platforms, offering interfaces to many of their specific features.

### 4.2 Implementation of AES Algorithms

The three AES algorithms *OriginalAES, TableLookupAES* and *HardwareAES* are implemented for TinyOS as user-defined components. A unified interface is used for each of the variants as shown in Figure 5.

**Figure 5:** Interfaces of AES implementations

```
interface OriginalAES {
    command void AES_set_encrypt_key(const uint8_t *key);
    command void AES_encrypt(uint8_t *in, uint8_t *out);
    command void AES_decrypt(uint8_t *in, uint8_t *out);
}

interface TableLookupAES {
    command void AES_set_encrypt_key(const uint8_t *key);
    command void AES_encrypt(uint8_t *in, uint8_t *out);
    command void AES_decrypt(uint8_t *in, uint8_t *out);
}

interface HardwareAES {
    command void AES_set_encrypt_key(const uint8_t *key);
    command void AES_encrypt(uint8_t *in, uint8_t *out);
}
```

The *OriginalAES* implementation is based on Niyaz's AES Implementation in C/C++ (Niyaz, 2009), which follows the original AES algorithm. Some modifications have been incorporated to adopt it to the sensor node environment, such as reducing memory moves, using global variables and restricting the implementation to support 128-bit keys only. Further, the source code is also modified to comply with the NesC format, containing components, modules, and interfaces. The whole procedure is divided into a key setup process, an encryption process and a decryption process. The command *AES_set_encrypt_key* is used to derive the used round keys from the 128-bit key. The commands *AES_encrypt* and *AES_decrypt* can only be called after *AES_set_encrypt_key*, as otherwise the round keys are not correctly initialised. These commands encrypt/decrypt the content at the parameter *in* and store the result at the parameter *out*.

As discussed in Section 3, AES can be optimized by replacing some operations with table lookups. The *TableLookupAES* implementation is modified to comply with NesC format and the ATmega128L hardware. Further, optimisations based on the recommendations published by Bertoni et al. (2003), Zhang and Niu (2008) and Diala, Ault, and Bagchi (2008) are incorporated. The same interface and functions as for the *OriginalAES* are used.

The *HardwareAES* implementation uses the CC2420's standalone AES encryption mode, as it is the only mode that does not involve radio communication or other extra features such as authentication. Analogous to the software implementations, the command *AES_set_encrypt_key* initialises the round keys. Since the key expansion computation is done by hardware, the *Hardware_AES* implementation's key setup process involves setting security control registers to proper values and writing the 128-bit encryption key to KEY RAM in the CC2420 chip. The command *AES_encryt* copies the content at the parameter "*in*" to the SABUF RAM (specially designed RAM for storing a single plaintext block for encryption purposes) and starts the encryption. When the encryption process is finished, the content in SABUF RAM is replaced with the encrypted data. The *AES_encryt* is copying the encrypted data to the parameter "*out*".

All three implementations were verified for correctness by matching their encryption/ decryption results against the samples presented in FIPS-197.

### 4.3 Performance and Power Consumption Measurement

Test applications have been developed to analyse execution times, memory usage and power consumption of the key-setup phase and the encryption/decryption phases of the AES implementations. To increase the accuracy of the measurements, these applications repeat each operation 100 times. The resulting average values are then scaled down to a single operation value.

Measurements of the ROM and RAM usages are provided by TinyOS, which provides a function to display memory usage after successful compilation. The execution time and power consumption are measured with the Agilent 66321D and 14565B. The 66321D supplies the power to the MicaZ node and records the voltage and current supply/usage. The 14656B processes these records and provides corresponding graphs.

## 5. Analysis Results

This section discusses the results obtained from the performance and power consumption analysis performed as outlined in Section 4. For all comparative figures, the values of *OriginalAES* are taken as the baseline.

### *5.1 Memory Usage*

The memory usage for the key setup procedure of the three implementations is summarised in Table 1. There are only minor differences in ROM usage between the OriginalAES and TableLookupAES implementations, while *HardwareAES* requires more than twice the amount of ROM. On the other hand, *HardwareAES* uses the least amount of RAM (about 61% of *OriginalAES*). This is as expected, as most operations are provided by the hardware support of the CC2420 chip. For the software implementations *TableLookupAES* requires only about 77% of the RAM of *OriginalAES.*

**Table 1:** Rom and Ram usage for Key Setup

| Implementation | *ROM (byte)* | *RAM (byte)* |
|---|---|---|
| *OriginalAES* | 7138 (100%) | 1322 (100%) |
| *TableLookupAES* | 6992 (98%) | 1014 (77%) |
| *HardwareAES* | 15282 (214%) | 801 (61%) |

Memory usage for encryption is summarised in Table 2, which are similar to key setup: *HardwareAES* uses the largest amount of ROM (156%), while *TableLookupAES* uses the least (87%) and *HardwareAES* uses the least amount of RAM (88%) while *TableLookupAES* requires the most (165%). Note that part of the higher ROM requirements for *HardwareAES* is due to the code to activate the CC2420 chip. As practical WSN applications need to activate the chip also for communication purposes, the ROM usage of applications using any of the software AES implementations would have a ROM requirement comparable to the hardware supported implementation.

**Table 2:** Rom and Ram usage for Encryption

| Implementation | *ROM (byte)* | *RAM (byte)* |
|---|---|---|
| *OriginalAES* | 9878 (100%) | 946 (100%) |
| *TableLookupAES* | 8642 (87%) | 1558 (165%) |
| *HardwareAES* | 15373 (156%) | 833 (88%) |

Table 3 depicts the memory usage for decryption. It can be seen, that there are only minimal differences in the usage of memory between the two software implementations.

**Table 3:** Rom and Ram used for Decryption

| Implementation | *ROM (byte)* | *RAM (byte)* |
|---|---|---|
| *OriginalAES* | 9320 (100%) | 2392 (100%) |
| *TableLookupAES* | 9496 (102%) | 2326 (97%) |
| *HardwareAES* | n/a | n/a |

## *5.2 ExecutionTime*

A summary of the measured execution time for the key setup procedure of the three AES implementations is presented in Table 4. For the two software implementations, the key setup procedure includes the expansion of the key into the keys required for each round, whereas for *HardwareAES* the procedure only includes copying the key to the CC2420's memory and preparation of several registers on the chip. The expansion of the key is performed during the encryption process itself.

**Table 4:** Execution Times for Key Setup

| Implementation | *Time (µs)* | *Ratio* |
|---|---|---|
| *OriginalAES* | 317.4 | 100% |
| *TableLookupAES* | 441.4 | 139.1% |
| *HardwareAES* | 224.5 | 70.7% |

It can be seen, that *HardwareAES* uses the least amount of time (70.7%). Among the software implementations, *OriginalAES* (100%) outperforms *TableLookupAES* (139.1%). The additional time in *TableLookupAES* is spent on preparing the employed tables.

Table 5 summarises the measured execution times for the three AES implementations. As expected, *HardwareAES* outperforms the software implementations.

**Table 5:** Execution Times for Encryption

| Implementation | *Time (µs)* | *Ratio* |
|---|---|---|
| *OriginalAES* | 1104.3 | 100% |
| *TableLookupAES* | 885.8 | 80.2% |
| *HardwareAES (a)* | 33.2 | 3.0% |
| *HardwareAES (b)* | 350.6 | 31.7% |

If only the time required for encryption is considered, then *HardwareAES* requires only 3% of the time of *OriginalAES* (entry *HardwareAES (a)* in Table 5). This result is in line with previously published work (Healy et al., 2007, Diala et al., 2008). However, to encrypt data with hardware support, the plaintext must be copied to the CC2420 chip and the resulting ciphertext must be copied back. The software implementations do not require moving the data, as the encryption process can directly accesses the plaintext in memory. These measurements reveal that the copying of data between the MicaZ's RAM and the CC2420 SABUF RAM takes a significant amount of time. The total execution time of *HardwareAES*, when including the data transfer, raises to 350.6μs (cf. entry *HardwareAES (b)* in Table 5): 151.8μs are required to copy the plaintext from RAM to the CC2420, 33.2μs are spend on encryption and 164.9μs are used to copy the resulting ciphertext back from the CC2420 to RAM. As a result, *HardwareAES* is only about 3 times faster than *OriginalAES* and only about 2.5 times faster than *TableLookupAES*.

## *5.3 Energy Consumption*

The Agilent 6321D Mobile Communications DC Source and the Agilent 14565B Device Characterization Software are used to monitor the power consumption of the three AES implementations. For the key setup procedure, the measurements are summarised in Table 6 and Figure 6 depicts the taken current measurement, where a voltage of 3V is used. Measurements taken are for 100 operations, which are then scaled down to the shown single-operation values. It can be seen, that while *HardwareAES* performs faster than the software implementations, it requires more than 3.5 times the current.

**Table 6:** Energy Consumption of Key Setup

| Implementation | *Time (ms)* | *Current (mA)* | *Energy (μJ)* | *Ratio* |
|---|---|---|---|---|
| *OriginalAES* | 0.3174 | 7.075 | 6.74 | 100% |
| *TableLookupAES* | 0.4414 | 7.162 | 9.48 | 141% |
| *HardwareAES* | 0.2245 | 25.414 | 17.12 | 254% |

Overall, the key setup procedure of *HardwareAES* consumes more than 2.5 times the energy of *OriginalAES* and nearly double the energy of *TableLookupAES*. The additional energy consumption of *HardwareAES* is due to the power required to power the CC2420 chip.

Similar results are obtained, as shown in Table 7 and Figure 7, for the energy consumption of encryption. While *HardwareAES* is the fastest implementation, it uses more energy than either software implementation (14% more than *OriginalAES* and about 40% more than *TableLookupAES)*. However, for encryption *TableLookupAES* is the most energy-efficient implementation, as it uses nearly 20% less energy than *OriginalAES*.

**Figure 6:** Current Measurement of Key Setup

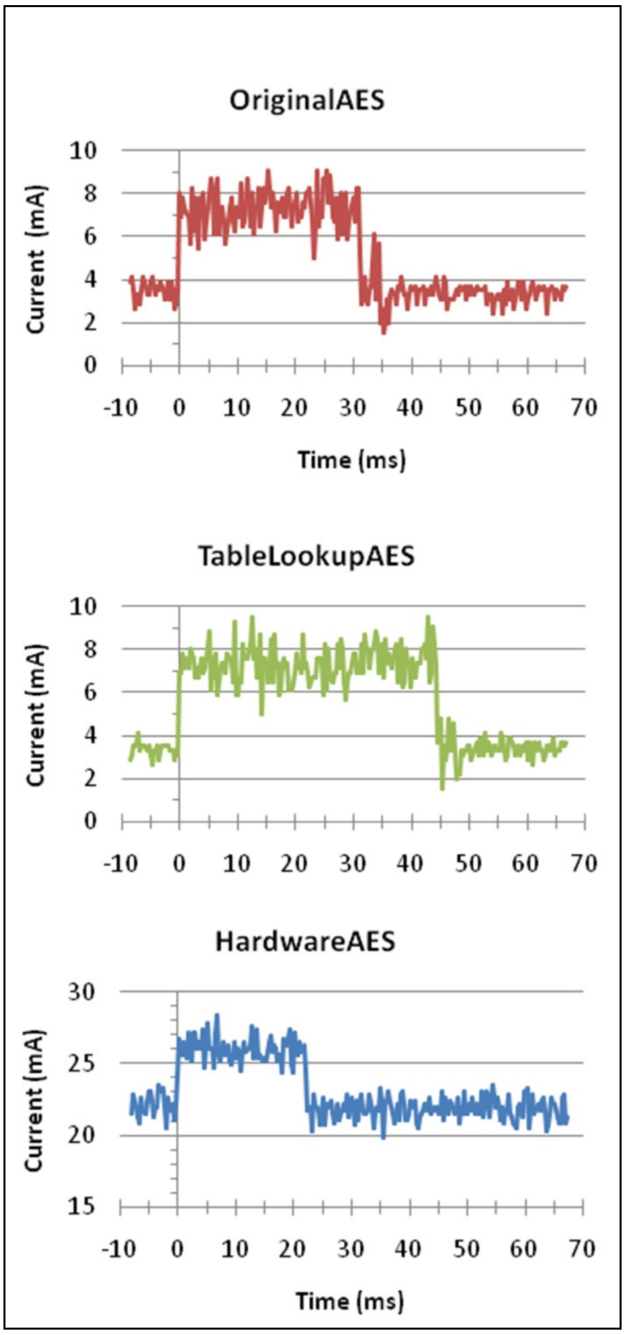
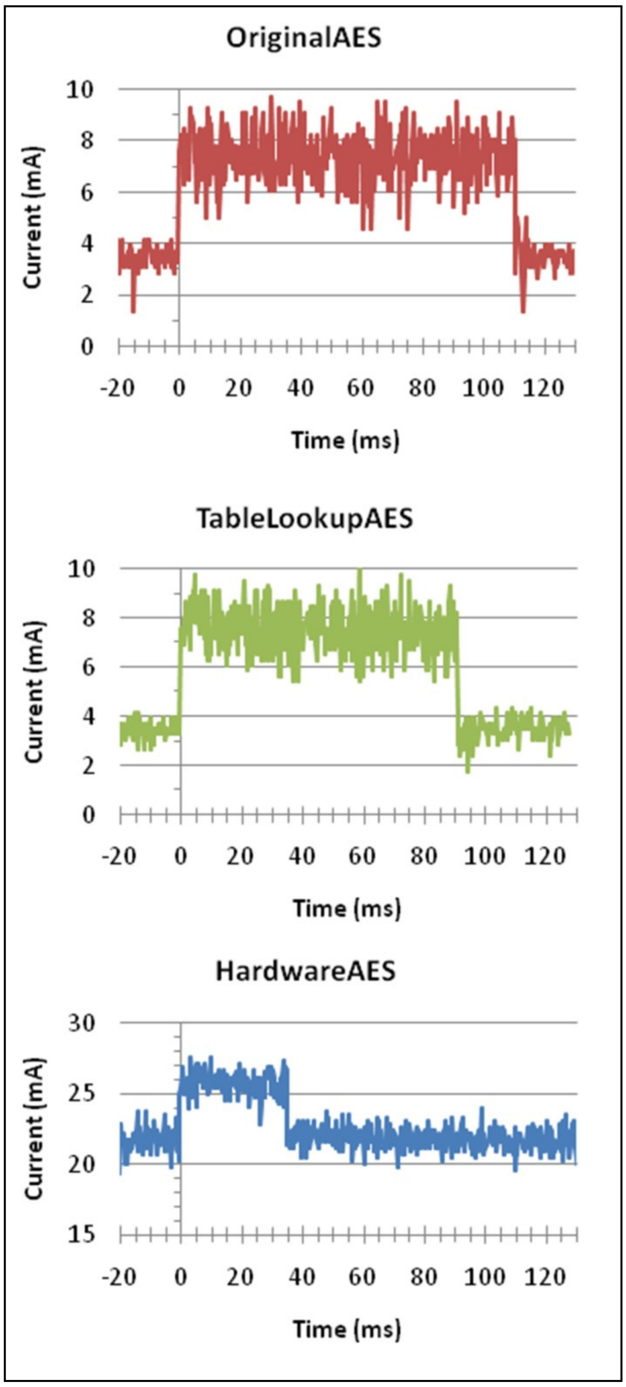


**Table 7:** Energy Consumption of Encryption

| Implementation | *Time (ms)* | *Current (mA)* | *Energy (μJ)* | *Ratio* |
|---|---|---|---|---|
| *OriginalAES* | 1.1043 | 7.116 | 23.57 | 100% |
| *TableLookupAES* | 0.8858 | 7.211 | 19.16 | 81% |
| *HardwareAES* | 0.3506 | 25.501 | 26.82 | 114% |

**Figure 7:** Current Measurement of Encryption



The overall relative energy consumption for a full encryption process (key setup and encryption) of the three AES implementations is shown in Table 8. The second column assumes a single encryption process per key setup, while the third column assumes that each encryption key is used for many encryption processes.

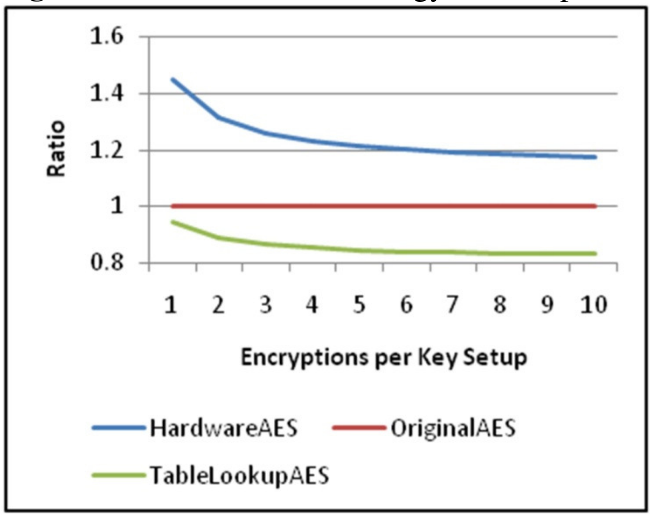
Figure 8 depicts the dependency of the relative energy consumption on the frequency of key changes. In the case that the key is changed for every encryption process, *HardwareAES* uses 45% more energy than *OriginalAES*, while *TableLookupAES* uses only 94.5% of *OriginalAES*.

**Table 8:** Overall Relative Energy Consumption

| Implementation | *Single Encryption per Key Setup* | *Many Encryptions per Key Setup* |
|---|---|---|
| *OriginalAES* | 100% | 100% |
| *TableLookupAES* | 94.5% | 81.3% |
| *HardwareAES* | 145% | 113.8% |

As the number of encryptions per key setup increases, the relative energy efficiency of *HardwareAES* and *TableLookupAES* improves. In the limit, *HardwareAES* requires about 13.8% more energy as *OriginalAES*, while the energy consumption of *TableLookupAES* decreases to 81.3% of *OriginalAES*.

**Figure 8:** Overall Relative Energy Consumption



## 6. Conclusions

In this paper a comparative performance and energy consumption analysis of *OriginalAES* (exclusively software AES using the original Rijndael algorithm), *TableLookupAES* (exclusively software AES using an optimised table lookup AES) and *HardwareAES* (hardware supported AES using the Chipcon CC2420 RF transceiver chip) implemented on a MicaZ sensor node has been presented.

Analysis of the memory requirements of the three implementations shows that they are all suitable for the MicaZ sensor node. As expected, investigation of execution times reveals that hardware supported encryption works faster than the pure software implementations. However, a significant portion of the speed advantage of hardware support is lost due to extra memory accesses required by *HardwareAES*. As a result, the key setup procedure is only about 30% faster than the fastest software implementation and encryption is only about 2.5 times faster than the best software implementation.

However, for most WSN applications the lifetime of the network relates deeply to the lifetime of the sensor node's power supply. Thus, energy consumption – rather than execution time - is often the decisive feature when a choice of multiple solutions is available. The presented analysis revealed that the speed advantage of hardware support is contrasted by its increased energy consumption. The key setup procedure of *HardwareAES* uses more than 2.5 times the energy than *OriginalAES* and nearly double the energy of *TableLookupAES*. Further, the encryption process of *HardwareAES* uses 14% more energy than *OriginalAES* and about 40% more than *TableLookupAES*. If the entire process, consisting of key setup and encryption is considered, the results depend on the relative frequency of encryptions per key setup: If the key is changed for every encryption process, *HardwareAES* is quite inefficient, as it uses 45% more energy than *OriginalAES* while *TableLookupAES* uses 94.5% of *OriginalAES*. As the number of encryptions per key setup increases, the relative efficiency of *TableLookupAES* and *HardwareAES* increases to 81.3% and 113.8%, respectively. Overall, while *HardwareAES* is the fastest implementation, it requires more energy than the software implementations. In conclusion, *TableLookupAES* is the most energy efficient AES implementation. Consequently, a node's lifetime can be significantly extended by choosing this implementation.

## References

Agilent (2007) "66321D Mobile Comm DC Source w/ Battery Emulation, DVM Datasheet," Obtained through Internet: http://www.home.agilent.com/agilent/product.jspx?cc=US&lc=eng&nid=-35489.536881812, [accessed 04.04.2010]

Akyildiz, I.F., Su, W., Sankarasubramaniam, Y. and Cayirci, E. (2002) "Wireless sensor networks: a survey," *Computer Networks*, Vol. 38, No. 4, March 2002, pp. 393-422.

Bertoni, G., Breveglieri, L., Fragneto, P., Macchetti, M. and Marchesin, S. (2003) "Efficient Software Implementation of AES on 32-Bit Platforms," *Lecture Notes in Computer Science*, Vol. 2523, January 2003, pp. 129-142.

Borms, J., Steenhaut, K., Phung, K.H. and Lemmens, B. (2010) "A traffic-adaptive multi-channel MAC protocol for wireless sensor networks", Proceedings of International Conference on Communications and Electronics, 2010

Chipcon, (2004) "CC2420 datasheet", *Chipcon*, Obtained through Internet: http://www.chipcon.com/files/CC2420_Data_Sheet_1_3.pdf [accessed 04.04.2010]

Chou, P. H. and Park, C. (2005) "Energy-efficient platform designs for real-world wireless sensing application," *Proceedings International Conference on Computer-aided design*, San Jose, November, 2005, pp. 913-920.

Courtois, N. and Pieprzyk, J. (2002) "Cryptanalysis of Block Ciphers with Overdefined Systems of Equations," *Lecture Notes in Computer Science* , Vol. 2501, pp. 267–287. 2002

CrossBow (2010), "MICAZ Datasheet," *CrossBow*, Obtained through Internet: http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf, [accessed 04.04.2010]

Daemen, J. and Rijmen, V. (1999) "AES Proposal: Rijndael", 1999.

Diala, S., Ault, A. and Bagchi, S. (2008) "Optimizing AES for embedded devices and wireless sensor networks", *Proceedings of the 4th International Conference on Testbeds and research infrastructures for the development of networks & communities*, Innsbruck, March 2008.

FIPS 197 (2001) “Advanced Encryption Standard FIPS-197,” Federal Information Processing Standards, Obtained through Internet: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf [accessed 04.04.2010]

Galindo, D., Roman, R. and Lopez, J. (2008) “A Killer Application for Pairings: Authenticated Key Establishment in Underwater Wireless Sensor Networks”, Lecture Notes in Computer Science, 2008, Volume 5339/2008, pp.120-132

Gay, D., Levis, P., Behren, R.V., Welsh, M., Brewer, E. and Culler, D. (2003) “The nesC Language: A Holistic Approach to Networked Embedded Systems”, *Proceedings of ACM Conference on Programming Language Design and Implementation (PLDI)*, San Diego, May 2003, pp. 1-11.

Govil, K., Chan, E., and Wasserman, H. (1995) “Comparing algorithms for dynamic speed-setting of a low-power CPU,” *Proceedings of ACM MobiCom’95*, Berkeley, CA, November 1995, pp. 13–25.

Hathaway, L. (2003) “National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information,” *Committee on National Security Systems*, June 2003. Obtained through Internet: http://www.cnss.gov/Assets/pdf/cnssp_15_fs.pdf [accessed 04.04.2010]

Healy, M., Newe, T., and Lewis, E. (2007) “Efficiently securing data on a wireless sensor network,” *Journal of Physics: Conference Series*, Vol. 76, No. 1, 2007.

Karlof, C., Sastry, N. and Wagner, D, (2004) “TinySec: a link layer security architecture for wireless sensor networks,” *Proceeding 2nd International Conference on Embedded Networked Sensor Systems*, Baltimore, April 2004, pp 162-175.

Kohno, E., Ohta, T. and Kakuda, Y. (2009) “Secure decentralized data transfer against node capture attacks for wireless sensor networks”, Proceedings of International Symposium on Autonomous Decentralized Systems, 2009.

Law, Y.W., Doumen, J. and Hartel, P. (2006) “Survey and benchmark of block ciphers for wireless sensor networks,” *ACM Transaction on Sensor Networks*, Vol. 2, No. 1, February 2006, pp. 65-93.

Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E. and Culler, D. (2005) “TinyOS: An Operating System for Sensor Networks,” *Ambient Intelligence, Part II*, Heidelberg, 2005, pp. 115-148.

Lorch, J. and Smith, A. (1996) “Reducing processor power consumption by improving processor time management in a singleuser operating system,” *Proceedings of ACM MobiCom’96*, New York, 1996, pp. 143-154.

Min, R., Furrer, T. and Chandrakasan, A. (2000) “Dynamic voltage scaling techniques for distributed microsensor networks,” *Proceedings of the IEEE Computer Society Annual Workshop on VLSI (WVLSI'00)*, Orlando, April 2000, pp. 43.

NIST, (1998) “Skipjack and KEA Algorithm Specifications Version 2.0,” 1998. http://www.jya.com/skipjack-spec.htm, 04.04.2010

Niyaz, P.K., (2009), “Advanced Encryption Standard (AES) Implementation in C/C++ with comments”. Obtained through Internet: http://www.hoozi.com/post/829n1/advanced-encryption-standard-aes-implementation-in-c-c-with-comments-part-1-encryption, [accessed 15.09.2009]

Olteanu, A., Xiao, Y. and Zhang, Y. (2009) "Optimization between AES Security and Performance for IEEE 802.15.3 WPAN," IEEE Transactions on Wireless Communications, Vol. 8, Nov. 12, Dec. 2009, pp. 6030-6037

Olteanu, A. and Xiao, Y. (2010) "Security Overhead and Performance for Aggregation with Fragment Retransmission (AFR) in Very High-Speed Wireless 802.11 LANs," IEEE Transactions on Wireless Communications, Vol. 9, No. 1, Jan. 2010, pp. 218-226.

Raghunathan, V., Schurgers, C., Park, S. and Srivastava, M. (2002) “Energyaware wireless sensor networks,” *IEEE Signal Processing* Vol. 19, No. 2, 2002, pp. 40-50.

Rivest, R. (1995) “The RC5 encryption algorithm,” *Proceeding Leuven Workshop on*

*Fast Software Encryption*, *Springer-Verlag*, Leuven, 1995, pp 86-96.

Romer, K. and Mattern, F. (2004) “The design space of wireless sensor networks,” *IEEE Wireless Communications*, Volume 6, No. 11, December 2004, pp. 54-61.

Shih, E., Cho, S., Ickes, N., Min, R., Sinha, A., Wang, A. and Chandrakasan, A. (2001) “Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks,” *Proceedings of ACM MobiCom’01*, Rome, Italy, July 2001, pp. 272–286.

Sohraby, K., Minoli, D. and Znati, T.F. (2007) “Wireless sensor networks: technology, protocols, and applications,” John Wiley & Son, 2007.

Vitalettim, A. and Palombizio, G. (2007) “Rijndael for sensor networks: is speed the main issue?” *Electronic Notes in Theoretical Computer Science*, Vol. 171, No. 1, April 2007, pp. 71-81.

Xiao, Y., Chen, H., Sun, B., Wang, R. and Sethi, S. (2006a) “MAC Security and Security Overhead Analysis in the IEEE 802.15.4 Wireless Sensor Networks,” EURASIP Journal on Wireless Communications and Networking, vol. 2006, Article ID 93830, 12 pages, 2006.

Xiao, Y., Sun, B., Chen, H., Guizani, S. and Wang, R. (2006b) "Performance Analysis of Advanced Encryption Standard (AES)," Proc. of GLOBECOM 2006,

Xiong, X., Wong, D. S., and Deng, X. (2009) “TinyPairing: Computing Tate Pairing on Sensor Nodes with Higher Speed and Less Memory”, Proceedings of IEEE International Symposium on Network Computing and Applications, 2009.

Yick, J., Mukherjee, B. and Ghosal, D. (2008) “Wireless sensor network survey,” *Computer Networks*, Vol. 52, No. 12, August 2008, pp. 2292-2330.

Zhang, F. and Niu, Y. (2008) “Rijndael Arithmetic Analyse and Optimize”, Proceedings of International Conference on Wireless Communications, Networking and Mobile Computing, 2008.