Analysis of Hardware Encryption Versus Software Encryption on Wireless Sensor Network Motes

Chapter · January 2008

DOI: 10.1007/978-3-540-79590-2_1

3 authors, including:

Thomas Newe
University of Limerick
102 PUBLICATIONS **480** CITATIONS
SEE PROFILE

Elfed Lewis
University of Limerick
320 PUBLICATIONS **1,589** CITATIONS
SEE PROFILE

Analysis of Hardware Encryption versus Software Encryption on Wireless Sensor Network Motes

Michael Healy, Thomas Newe and Elfed Lewis

Optical Fibre Sensors Research Centre,
Department of Electronic and Computer Engineering,
University of Limerick,
Limerick,
Ireland.
{michael.healy, thomas.newe, elfed.lewis}@ul.ie

Abstract Due to the sensitive and often personal nature of sensor data that many wireless sensor networks collect, the security of this data must be guaranteed. This is fast becoming an important concern for sensor networks which are finding applications in the military and home health domains. The best and often the only way to secure this data is to encrypt it using a secure encryption algorithm before it is transmitted over the air ways. Due to the constrained nature of the resources, memory and clock speeds, available on sensor nodes however, the cost, both in terms of power consumption and speed of encryption, of a software based encryption procedure can often outweigh the risks of the transmission being intercepted. This paper presents a solution to reduce this cost of employing encryption by taking advantage of a resource already available on many sensor nodes, including the Crossbow MICAz and MoteIV's TmoteSKY; this resource being the AES encryption module available on the Chipcon CC2420 transceiver chip. The performance of using this method of securing data on a sensor network against using software implementations of some of the most popular cipher algorithms suitable for WSN is then analysed for both hardware platforms.

Keywords sensor data security, wireless sensor networks, AES, CC2420

1 Introduction

Technological advancements in recent years have enabled the development of tiny, cheap disposable and self contained battery powered computers, known as sensor nodes or "motes", which can accept input from an attached sensor, process this input and transmit the results wirelessly to some interested device(s). When a number of these nodes work to-

gether, conceivably up to hundreds of thousands, a Wireless Sensor Network (WSN) is formed.

Research in the area of wireless sensor networks has become increasingly widespread in recent years, partly due to their wide range of potential uses and also partly due to the fact that the technology enabling such networks is now widely available from many different suppliers, such as: Crossbow, MoteIV, Intel and SUN (java based motes).

These wireless sensor networks have the potential to allow a level of integration between computers and the physical world that, to date, has been virtually impossible. The uses for such networks is almost limitless and include such diverse applications as a counter sniper system for urban warfare [1] tracking the path of a forest fire [2], determining the structural stability of a building after an earthquake [3], or tracking people or objects inside a building [4], etc.

Advances in wireless communication have been a major factor in allowing the development of large networks of sensors. However, as stated in the IEEE 802.15.4 standard specification [5], the wireless connectivity of the sensors is not so much a feature of the sensors but rather an application enabler (unlike the majority of wireless applications currently available). This is the case because wired sensor networks on the required scale would be very costly to build and maintain and also very costly to install, making them impractical for many application areas, such as: environmental monitoring, home health and military to list but a few.

Despite making such sensor networks possible, the wireless nature of the sensors presents a number of problems to the developer. Chief among these is the problem of security. Many WSNs are designed to collect data which is sensitive in some way, such as information about a person's health, confidential data about a company's manufacturing process, troop locations on a battlefield, etc. Without adequate security this data can easily be read and/or modified by an attacker. Due to the wireless nature of the sensor networks detecting and preventing eavesdropping and modification of the data is greatly complicated. Also the constrained nature of resources on the wireless sensor nodes means that security architectures used for traditional wireless networks are not viable.

As power and RAM are normally the most constrained resources on a wireless node, a memory and power efficient cryptographic algorithm is a very important part of any proposed security architecture for a WSN. Traditionally RC5 [6] and Skipjack [7] are considered to be the most suitable algorithms for WSNs [8, 9] due to their speed and memory usage but as shown by Law et al [10] AES [11] or MISTY1 [12] can be more suitable, depending on the situation.

A solution to reduce power and memory costs, as well as greatly increasing speed of encryption, is to use a hardware implementation of a cipher. As AES is currently the most widely used and secure block cipher available it has a larger range of commercial implementations than for any other algorithm. However none of the currently available wireless sensor nodes has such a chip already included and adding one can prove difficult and very time consuming, especially for large deployments. Another solution is available however. As can be seen by table 1 the majority of the currently available sensor nodes, both those commercially available and those created by research institutions, use the same IEEE 802.15.4 compliant transceiver, the Chipcon CC2420 [13]. The CC2420 features a number of security operations, including AES encryption. We test the performance of using this hardware encryption on the CC2420 compared with a software implementation of the AES algorithm and the TinySec [8] implementations of RC5 and Skipjack. Two platforms are used for this testing; Crossbow's MICAz and MoteIV's TmoteSKY.

Platform	MCU	RAM (KB)	Program Memory (KB)	Non-volatile Data Mem (KB)	Radio Chip
BTnode3	ATMega 128	64	128	180	CC1000/ ZV4002
Cricket	ATMega 128	4	128	512	CC1000
imote2	Intel PXA271	256	32,000	0	CC2420
mica2	ATMega 128	4	128	512	CC1000
mica2Dot	ATMega 128	4	128	512	CC1000
micaz	ATMega 128	4	128	512	CC2420
Shimmer	TI MSP430	10	48	Up to 2 GB	CC2420/ WML-C46
TelosA	TI MSP430	2	60	512	CC2420
TelosB	TI MSP430	10	48	1	CC2420
Tmote Sky	TI MSP430	10	48	1	CC2420
XYZ	ARM 7	32	256	256	CC2420

Table 1. Features of currently available Wireless Sensor Nodes [20]

2 Background

In this section the target hardware and the operating system that was used is described. An explanation of why AES is considered an appropriate choice for use in wireless sensor network data protection is also provided as well as a description of the block ciphers traditionally used in wireless sensor networks.

2.1 Sensor Node Hardware

The sensor nodes that work together to form a WSN are composed of four sub-systems; a computing sub-system, a communication sub-system, a power subsystem and a sensing sub-system.

The computing sub-system consists of a processor and memory, which includes program memory, RAM and possibly non-volatile data memory. An important aspect of processors in sensor nodes is different operational modes, usually Active, Idle and Sleep. This is important so as to preserve power as much as possible without impeding the operation of the processor when it is required.

The communication sub-system is required to enable the sensor nodes to communicate with each other and with a base station. Generally the communication sub-system is a short range radio but the use of infrared communication, ultrasound and inductive fields has also been explored. Most currently available sensor nodes use a radio chip which conforms to the IEEE 802.15.4 standard, but as can be seen from Table I some nodes use Bluetooth as an alternative.

The power sub-system consists of a battery which supplies power to the sensor node. Due to the long term unattended operation of nodes in a WSN the developer must ensure

every aspect of the network such as communication algorithms, localization algorithms, sensing devices, etc., must be as efficient as possible in their power usage. A power generator may also be included to recharge the battery onsite. Photovoltaic, motion/vibration and thermoelectric energy conversion are all possible sources of power, depending on the location of the node [15].

Sensor transducers translate physical phenomena to electrical signals. Therefore the sensing sub-system of the node is its link to the outside world. The output of the sensors may be digital or analog signals. If the output is analog the node must also include an Analog to Digital Converter (ADC) in order to allow the processor to read the data. Some nodes have sensors built in, but many do not, instead providing suitable ports allowing a variety of sensors to be attached for more versatility.

As already mentioned the CC2420 transceiver chip from Chipcon is currently the most popular radio chip on wireless sensor nodes. The CC2420 works in the 2.4Ghz band, is IEEE 802.15.4 compliant, ZigBee ready, low cost, and designed for low-voltage and low-power wireless applications. These features, along with a host of other features make the CC2420 ideal for use on sensor nodes.

The CC2420 feature that is most interesting is its ability to perform hardware security operations. The CC2420 is capable of performing IEEE 802.15.4 MAC security operations, including counter (CTR) mode encryption and decryption, CBC-MAC authentication and CCM encryption plus authentication. The CC2420 also offers plain stand-alone encryption of 128 bit blocks. Each of these security functions are based on AES encryption using 128 bit keys.

One limitation of the CC2420 security wise is that it does not offer AES decryption. However the impact of this limitation is negated by the fact that for many applications any decryptions are performed by the base station which can be a much more powerful device and/or has less limitation on its power supply. Another option is to use a block cipher mode of operation which does not require a decryption function, such as Cipher Feed Back (CFB) mode, Output Feed Back (OFB) mode, or Counter (CTR) mode.

2.2 TinyOS

As can be seen from table 1 [20] the specifications of the target hardware for the tested system are varied. The variety of processors, radio chips as well as the varying amount of RAM and program memory means the software written for each node would need to be significantly different. However a number of operating systems already exist to solve this problem. The TinyOS [16] operating system was chosen as the development platform, mainly because it currently supports the largest range of hardware and has good power management [17].

TinyOS was the first operating system specifically designed for WSNs and as a result it still has the largest user base and is the standard by which the other operating systems are judged. TinyOS was initially developed at the University of California, Berkeley but is now being developed by a consortium and is open source, making it easy for developers to customize it as required.

TinyOS is not an operating system in the traditional sense; rather it is a programming framework which contains a set of components that allows an application specific OS to be constructed for each particular need, consisting of selected system components and custom components. The principal goal of this component based architecture is to allow application designers to build components that can easily work together to produce a complete, concurrent system but still performs extensive compile time checks.

2.3 TinySec

TinySec [8] is a link layer security architecture for wireless sensor networks, implemented for the TinyOS operating system. In order to overcome the processor, memory and energy constraints of sensor nodes TinySec leverages the inherent sensor network limitations, such as low bandwidth and relatively short lifetime for which the messages need to remain secure, to choose the parameters of the cryptographic primitives used. Using these parameters TinySec adds less than 10% energy, latency and bandwidth overhead.

TinySec has two modes of operation: authenticated encryption (TinySec-AE) and (TinySec-Auth). With authenticated encryption, TinySec encrypts the data payload and authenticates the packet with a Message Authentication Code (MAC). The MAC is computed over the encrypted data and the packet header. In authentication only mode, TinySec authenticates the entire packet with a MAC, but the data payload is not encypted.

An important feature of TinySec is its ease of use, as many application developers will either implement the security features incorrectly or leave out any security entirely if the security API is difficult to use. TinySec solves this problem by integrating into TinyOS at a low level. To enable TinySec-Auth mode, only one line needs to be added to the application's Makefile, the application code itself does not need to be touched. TinySec-Auth mode is the default mode of operation for TinySec. To enable TinySec-AE mode the TinySecC configuration file needs to be wired into the application and one function call needs to be given.

All of TinySec's security primitives are built on top of the employed block cipher, and currently users can choose between RC5 of Skipjack.

2.4 Block Ciphers

The block cipher employed is probably the most important cryptographic building block to get right for securing a wireless sensor network. A balancing act between security, memory requirements and power consumption is required. Obviously the cipher needs to be secure as if it is easily broken there is no point in using it, the data might as well be sent unencrypted. However, if the cipher used takes up so much program memory, RAM or energy as to seriously impede the sensing, data processing and data transmission functions or the effective lifetime of the sensor node, the use of the cipher becomes impractical.

An important point to consider when determining the cipher to be used is that the length of time the data needs to be kept secure in a wireless sensor network is often much shorter than in a traditional wireless network. As a result, a less secure, but more memory and energy efficient cipher can often be more appropriate.

2.4.1 AES/Rijndael

Rijndael [11] was chosen as the Advanced Encryption Standard (AES) by the National Institute of Standards and Technology of the United States in 2001 to replace the existing Data Encryption Standard (DES). It is also one of the ciphers recommended for use by Japans CRYPTREC and by the New European Schemes for Signature, Integrity and Encryption (NESSIE) consortium.

Rijndael is designed to work with three different key lengths of 128 bits, 192 bits or 256 bits depending on security requirements. A key length of 128 bits is considered more than sufficient for securing wireless sensor networks because the length of time any collected data needs to remain confidential will have long since expired before an exhaustive key search on this length key can be performed using currently available technology. However for increased security, at a cost of memory and speed, one of the longer key lengths can also easily be employed.

One of the criteria used to choose AES was the cipher's efficiency and performance on a variety of platforms, from 8-bit smart cards to high end processors. For this reason Rijndael should be well suited for use on wireless sensor nodes, despite their limited processing power.

Rijndael has come under intense scrutiny both during the review process before it was chosen as AES and also since then. As of January 2008 no feasible attacks against the algorithm have been published and so it is still considered secure.

2.4.2 RC5

RC5 [6] is a block cipher designed by Ronald Rivest in 1994 and is noted for its simplicity. RC5 is also noted for its flexibility, with a variable block size of 32, 64 or 128 bits, variable key size of 0 to 2040 bits and a variable number of rounds of 0 to 255.

TinySec uses a block size of 64 bits, key size of 64 bits and 12 rounds. For traditional networks these parameters would no longer be considered secure because 12 round RC5 using 64 bit blocks is susceptible to a differential attack [19]. However for this attack 244 chosen plaintexts are required so it is highly unlikely to be an effective attack during the lifetime of any particular sensor network.

The key length of 64 bits is the biggest weakness of the TinySec implementation of RC5. Limiting the key to 64 bits means it would be possible to complete an exhaustive key search attack within the lifetime of some networks.

Despite being very fast and still sufficiently secure RC5 is not widely used due to the fact that it is a patented algorithm. Patent free alternatives with comparable properties are widely available.

2.4.3 Skipjack

Skipjack [7] was initially a classified algorithm developed by the National Security Agency in the USA. Since its declassification in 1998 Skipjack has come under intense scrutiny and is still considered secure.

Skipjack uses 64 bit blocks, an 80 bit key and has 32 rounds. While an 80 bit key is significantly more secure than the 64 bit key used in the TinySec's RC5 implementation, with current technology trends it is quite likely that this length key will be rendered insecure to an exhaustive key search attack within the next 5 years.

	Block Size	**Key Size**	**No. Rounds**
AES	128 bits	128 bits	10
RC5	64 bits	64 bits	12
Skipjack	64 bits	80 bits	32

Table 2. Comparison of implemented block ciphers

3 Implementation

The secure application was implemented under TinyOS which employs the CC2420 hardware encryption features in order to encrypt 16 bytes of data at a time and then send this data to a listening base station. This application was then run on two of the most popular sensor node platforms which use the CC2420 radio, Crossbow's MICAz and MoteIV's Tmote SKY.

Using the stand alone encryption of the CC2420 is relatively straight forward. First off the two Security Control registers, SECCTRL0 and SECCTRL1, which control the security operations of the CC2420, need to be set. As we are using key 0 as our stand-alone key we can set both of these registers to 0. We then write the 128 bit key and the plaintext to the appropriate RAM locations on the CC2420; from location 0x100 and 0x120 respectively. The SAES command strobe is then sent in order to begin the encryption operation. We can tell when the encryption operation is completed by reading the ENC_BUSY status bit, which tells us whether the encryption module is busy or not. We read the CC2420's status bits by sending the SNOP command strobe. The final step is to read back the cipher text. As the encryption module overwrites the plaintext with the cipher text this is simply a case of reading 16 bytes from the location the plaintext was originally written to, i.e. from location 0x120. It is worth noting that the CC2420 RAM write operation also outputs the data currently at the location being written to, so that a new plaintext can be written at the same time as reading out the previous cipher text in order to streamline encrypting data longer than 128 bits.

It is also worth noting that when using the other security operations supported by the CC2420, i.e. the IEEE 802.15.4 MAC security modes, the encryption happens after the message is buffered in the TXFIFO buffer, and the decryption happens before the message is read out of the RXFIFO buffer. This means that the encrypted message does not have to be read from the CC2420 at any stage, which would only have to be sent back to the transceiver for another operation anyway, so using these modes is even simpler and more transparent for the application developer than using the stand-alone encryption mode.

The only difference between the versions of the application on each platform is that for the MICAz version the function to read from the RAM on the CC2420 chip had to be implemented and integrated whereas for the Tmote SKY this function was already made available by TinyOS.

For comparison purposes a number of software implementations of the AES encryption algorithm were also created. The first implementation is used for reference and is as simple as possible based on the Rijndael AES proposal [11]. The second software based AES version has some simple optimisations. These optimisations are primarily using pre-computed lookup tables and, as we are using a fixed block size and key size of 128 bits each (to match the parameters of the stand-alone encryption on the CC2420), we were able to unroll some loops.

The default TinyOS operation is to load the lookup tables into RAM at boot up. As this consumes a significant amount of RAM we created a second implementation which prevents TinyOS from doing this.

As already mentioned RC5 and Skipjack are the conventional algorithms used for wireless sensor networks. In order to compare the performance of the AES encryption on the CC2420 to the performance of these algorithms we implemented a TinyOS application which encrypts 16 bytes of data using each of these algorithms, these implementations being based on the highly optimised TinySec versions. Currently TinySec is only available for the MICA2 platform and TinySec's block ciphers are optimised for use on this platform, using inline assembly code to speed up the most common operations. However, as can be

seen in table 1, the MICA2 and MICAz nodes use the same microprocessor, the ATMega 128L, so this optimised code runs unmodified on the MICAz. The Tmote SKY's processor is a Texas Instruments MSP430, and as a result our RC5 and Skipjack implementations do not run on this platform.

As already mentioned the TinySec implementations of the RC5 and Skipjack encryption algorithms are highly optimised and as a result our versions of these algorithms do not offer the same level of security as our AES applications. This is the case because, as can be seen in table 2, a 64 bit key is used for RC5 and a 80 bit key is used for Skipjack, whereas a 128 bit key is used for AES, both the hardware and software versions.

	MICAz		**Tmote SKY**	
	ROM	**RAM**	**ROM**	**RAM**
CC2420 AES	10058	437	11872	407
AES software Ref.	11980	2844	13206	2811
AES software	12720	1915	13980	1833
AES software min RAM	12748	625	14200	887
RC5	11050	533	-	-
Skipjack	11696	451	-	-

Table 3. Program memory and RAM usage (bytes)

4 Results

MICAz	**Tmote SKY**
7.88137 ms	13.29879 ms

Table 4. Time to generate lookup tables

Table 3 shows the amount of program memory and RAM each implementation uses on each platform. As can be clearly seen the AES implementation that uses the CC2420 encryption module uses almost 1 KB less ROM than the nearest competitor on both platforms and also comes out top in terms of RAM usage. The version of Skipjack used minimized RAM usage, as is the case with the TinySec implementation, where the lookup tables that are required were placed in program memory, however, as will be seen this comes at a cost of performance.

The AES software reference version generates the reference tables required at runtime in order to save program memory. This technique can also be used to save RAM. However as we saved these generated tables in static global arrays, they only needed to be generated once so we did not reap the RAM saving benefits. The time it takes to generate these tables is shown in table 4. In order to create a software implementation of AES which optimizes

both RAM and program memory usage these tables would need to be generated every time they were required.

Tables 5 and 6 show the amount of time it takes to execute various parts of the code for each implementation. Table 5 shows the amount of time it takes each application to run the setup procedure before encryption can commence. For the CC2420 AES version this involves setting the setting the security registers and loading the key and plaintext into the CC2420's RAM and the time taken to read the resulting cipher text back is also included. For the other five versions the setup procedure involves performing any key expansion required. Overall the AES implementations came out top in this category with the hardware version being by far the quickest and the three software versions being significantly quicker than the RC5 or Skipjack implementations. For the five software based versions the setup procedure only needs to be run each time the encryption key changes but for the hardware version some of the tasks we have included in the time taken for the setup procedure need to be run for each 128 bit block being encrypted, i.e. writing the plaintext to the CC2420's RAM and reading back the cipher text. However as mentioned in section 3 these two operations can be performed at the same time so the impact of this is not significant.

Table 6 shows the amount of time it takes to actually perform the encryption routine on 16 bytes of data. It is in this category that the CC2420 AES implementation really shows its worth, being over 15 times faster than the RC5 algorithm, over 24 times faster than the Skipjack version, and nearly 49 times faster than either of the AES software based implementations on the MICAz.

On the Tmote SKY the hardware based AES version is only about 4 times faster than the software based AES versions. Also the Tmote SKY is nearly 15 times slower than the MICAz when using the same hardware to perform the encryption. Also, as seen in table 5, the CC2420 AES version's setup procedure on the Tmote SKY is very slow compared to that of the MICAz's. These timing differences cannot be adequately explained by the processor and clock differences between the two platforms and so we suspect that the difference is due to TinyOS's platform specific routines that read from and write to the RAM on the CC2420.

	MICAz	**Tmote SKY**
CC2420 AES	294.003 µs	2.06034 ms
AES software Ref.	652.15 µs	940.83 µs
AES software	565.37 µs	752.27 µs
AES software min RAM	574.85 µs	934.83 µs
RC5	1.68591 ms	-
Skipjack	687.674 µs	-

Table 5. Time to run up setup procedure

As seen in Tables 5 and 6, the RAM optimised software implementation of AES is slower than the un-optimised versions. This increase in processing time is due to the extra overhead of reading from the lookup tables placed in program memory over that of reading the same tables in RAM. This is a cost the Skipjack implementation also pays, which, as a

result, could be modified to have slightly improved execution times at a cost of more using more RAM if required.

AES was generally not considered viable for use on wireless sensor nodes because of its execution speed and RAM usage. However our RAM minimized software implementation of AES, which still has a lot of scope for more optimisation, only uses 92 bytes more RAM than an optimised RC5 implementation and is executed only about 2 times slower than an optimised Skipjack implementation. Therefore, as using the CC2420 to speed up encryption results in using at least 96 less bytes of RAM and executes at a minimum of 15 times faster than the alternatives discussed here, using this hardware aided method to secure data results in a significant boost in performance.

	MICAz	**Tmote SKY**
CC2420 AES	29.8310 µs	449.203 µs
AES software Ref.	1.49578 ms	1.94108 ms
AES software	1.45753 ms	1.87064 ms
AES software min RAM	1.54772 ms	1.94374 ms
RC5	465.685 µs	-
Skipjack	721.446 µs	-

Table 6. Time taken to encrypt 16 bytes

5 Conclusions

It has been shown that using hardware encryption instead of software based encryption is beneficial in terms of speed and memory usage for wireless sensor network motes. In particular we have shown that using a resource, the Chipcon CC2420 transceiver chip, already existing on two of today's most popular wireless sensor motes (Crossbow's MICAz and MoteIV's TmoteSKY) one can significantly reduce the cost of securing data on a sensor network with regards to power consumption and speed of encryption. Indeed it can be seen that the MICAz offers significant improvement over the TmoteSKY mote for hardware based encryption. The authors suspect that this is due to TinyOS's platform specific routines that read from and write to the RAM on the CC2420. This work is ongoing.

6 Acknowledgments

The authors wish to thank the following for their financial support:

- SFI Research Frontiers Programme grant number 05/RFP/CMS0071
- The Embark Initiative and Intel, who fund this research through the Irish Research Council for Science, Engineering and Technology (IRCSET) postgraduate Research Scholarship Scheme.

7 References

1. Á. Lédeczi, A. Nádas, P. Völgyesi, G. Balogh, B. Kusy, J. Sallai, G. Pap, S. Dóra, K. Molnár, M. Maróti, and G. Simon (2005) Counter sniper System for Urban Warfare. In ACM Transactions on Sensor Networks (TOSN), vol. 1, pp. 157-177.
2. C.-L. Fok, G.-C. Roman, and C Lu (2005) Mobile Agent Middleware for Sensor Networks: An Application Case Study. In Proc. 4th International Conference on Information Processing in Sensor Networks (IPSN'05), Los Angeles, California.
3. T. Schmid, H. Dubois-Ferrière, and M. Vetterli (2005) SensorScope: Experiences with a Wireless Building Monitoring Sensor Network. In Proc. Workshop on Real-World Wireless Sensor Networks (REALWSN'05), Stockholm, Sweden.
4. R. Want, A. Hopper, V. Falcão, and J. Gibbons (1992) The Active Badge Location System. In ACM Transactions on Information Systems (TOIS), vol. 10, pp. 91-102.
5. 802.15.4: Wireless Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs). (2003) New York: IEEE Standards Association.
6. R. Rivest (1994) The RC5 Encryption Algorithm. In Proc. 1994 Leuven Workshop on Fast Software Encryption, Leuven, Belgium.
7. NIST (1998) Skipjack and KEA Algorithm Specifications Version 2.0. NIST.
8. C. Karlof, N. Sastry, and D. Wagner (2004) TinySec: A Link Layer Security Architecture for Wireless Sensor Networks. In Proc. 2nd International Conference on Embedded Networked Sensor Systems, Baltimore, MD, USA.
9. A. Vitaletti, and G. Palombizio (2006) Rijndael for Sensor Networks: Is Speed the Main Issue?. In Proc. 2nd Workshop on Cryptography for Ah Hoc Networks, Venice, Italy.
10. Y. W. Law, J. Doumen, and P. Hartel (2006) Survey and Benchmark of Block Ciphers for Wireless Sensor Networks. In *ACM Transactions on Sensor Networks (TOSN)*, vol. 2, pp. 65-93.
11. J Daemen, and V. Rijmen (1999) AES Proposal: Rigndael.
12. M. Matsui (1997) New Block Encryption Algorithm MISTY. In Proc. 4th International Workshop on Fast Software Encryption, Haifa, Israel.
13. Chipcon (2004) CC2420 Datasheet [online], available: http://www.chipcon.com/files/CC2420_Data_Sheet_1_3.pdf [accessed 9 Feb 2007].
14. M. Healy, T. Newe and E. Lewis (2007) Resource Implications for Data Security in Wireless Sensor Network nodes. In Proc. International Conference on Sensor Technologies and Applications, Valencia, Spain.
15. S. Roundy, P. K. Wright, and J. M. Rabaey (2003) Energy Scavenging for Wireless Sensor Networks: With Special Focus on Vibrations, 1st ed: Springer.
16. J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister (2000) System Architecture Directions for Networked Sensors. In Proc. 9th International Conference on Architectural Support for Programming Languages and Operating Systems, Cambridge, Massachusetts, USA.
17. M. Healy, T. Newe, and E. Lewis (2007) Power Management in Operating Systems for Wireless Sensor Nodes. In Proc. IEEE Sensors Applications Symposium (SAS 2007), San Diego, California, USA.
18. K. Sun, P. Ning, and C. Wang (2006) TinySeRSync: Secure and Resilient Time Synchronization in Wireless Sensor Networks. In Proc. 13th ACM Conference on Computer and Communications Security, Alexandria, Virginia, USA.

19. A. Biryukov, E Kushilevitz (1998) Improved Cryptanalysis of RC5. In Proc. Advances in Cryptology---EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques. LNCS, vol. 1403. Springer-Verlag, 85--99.
20. M. Healy, T. Newe, and E. Lewis (2007) Securing Sensor Data on a Wireless Sensor Network. In Proc. 2nd International Conference on Sensing Technology, Palmerston North, New Zealand. pp 176-181. ISBN 978-0-473-12432-8.