R. Roman, C. Alcaraz, and J. Lopez, "A Survey of Cryptographic Primitives and Implementations for Hardware-Constrained Sensor Network Nodes", *Mobile Networks and Applications*, vol. 12, pp. 231-244, 2007.
http://doi.org/10.1007/s11036-007-0024-2
NICS Lab. Publications: https://www.nics.uma.es/publications

A Survey of Cryptographic Primitives and Implementations for Hardware-Constrained Sensor Network Nodes

Rodrigo Roman Cristina Alcaraz
Javier Lopez
Computer Science Department, University of Malaga, Spain
{roman,alcaraz,jlm}@lcc.uma.es

May 30, 2007

Abstract

In a wireless sensor network environment, a sensor node is extremely constrained in terms of hardware due to factors such as maximizing lifetime and minimizing physical size and overall cost. Nevertheless, these nodes must be able to run cryptographic operations based on primitives such as hash functions, symmetric encryption and public key cryptography in order to allow the creation of secure services. Our objective in this paper is to survey how the existing research-based and commercial-based sensor nodes are suitable for this purpose, analyzing how the hardware can influence the provision of the primitives and how software implementations tackles the task of implementing instances of those primitives. As a result, it will be possible to evaluate the influence of provision of security in the protocols and applications/scenarios where sensors can be used.

***Keywords* -** *Sensor Networks; Hardware; Cryptography.*

1 Introduction

One of the biggest challenges in the field of wireless sensor networks [1] is to provide the adequate security foundations for protocols and services. For this purpose, it is necessary that the nodes run cryptographic operations based on primitives such as *Symmetric Key Encryption* (SKE), *Hash functions*, and *Public Key Cryptography* (PKC). Without these primitives, it would not be possible to provide essential security services such as confidentiality of the communication channel, authentication of the peers involved in an information exchange, and integrity of the messages, among others.

Although the cryptographic primitives are usually complex, in terms of computational overhead and memory usage, the hardware resources available in the devices should be able to minimize their impact on the execution time of the secured services. However, the computing capabilities of a sensor node are very limited: a typical node has a 8Mhz microcontroller with less than 128kB of instruction memory and approximately 10kB of RAM memory. Therefore, it would be necessary to analyze how the existing primitives could perform over these highly-constrained nodes.

In order to explore the suitability of every primitive, that is, the suitability of their building blocks and internal operations, it is necessary to review the specific platforms of sensor nodes and their hardware components. With this knowledge, it can be possible to deduce which are the primitives

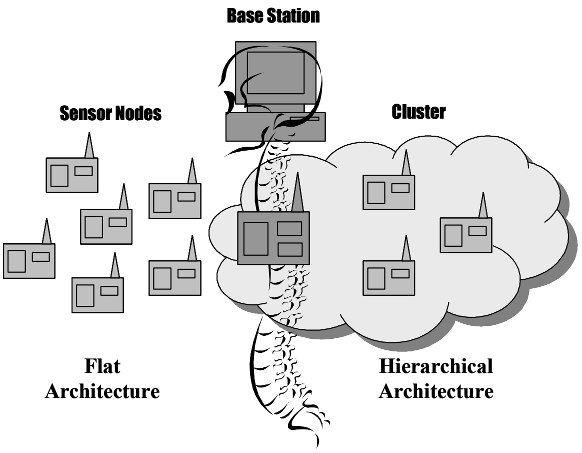

Figure 1: Overview of the Architecture of WSN

that could be more optimal for securing the network. Nevertheless, such primitives can be implemented in hardware or in software. It would be interesting to know whether a software implementation of a primitive is enough or not. Therefore, it is also important to examine the existent hardware and software implementations, being research work or available products, of the primitives. Achieving these goals is the purpose of this work.

The rest of this paper is organized as follows: first, an overview of the sensor network technology is presented, highlighting its vulnerabilities and the required primitives and services needed to assure the correct behavior of the network (section 2). Then, a description of the principal cryptographic primitives is introduced, alongside with their characteristics and modes of operation (section 3). Next, the principal elements of a sensor node, the microcontroller and the transceiver, are discussed, and the suitability of the primitives in these constrained hardware platforms is studied (section 4). Finally, both the hardware and software implementations of the primitives are shown (sections 5 and 6, respectively), comparing and analyzing their results. The conclusions of these analyses are presented in the last section (section 7).

2 Sensor networks and Security Requirements

2.1 Sensor Networks Overview

A wireless sensor network [1] is, roughly speaking, a group of highly-constrained hardware platforms called sensor nodes that collaborate towards a set of common goals. More specifically, those goals are monitoring (continuously monitor certain features of their surroundings), alerting (check whether certain physical circumstances are occurring, triggering an alarm), and provisioning of information "on-demand" (answer to a certain query regarding the properties of the environment or the network itself). Most of the functionality of a sensor network is data-driven, although it is also possible to use it as a distributed computing platform under specific circumstances.

All the functionality of the sensor network is provided thanks to the individual capabilities of the sensor nodes. A single sensor node has built-in sensors, limited computational capabilities, and communicates through a wireless channel. Therefore, they are able to get the physical information of their surroundings, process that raw information, and communicate with other nodes in its neighborhood. Nodes are also small in size, and can unobtrusively provide the physical information of any entity. Moreover, nodes are battery-powered, thus they can act independently and operate autonomously if required.

All the data created by the whole network must be accessed by a user, and this is where the base station operates. The base station is a more powerful and capable device that serves as an interface between the sensor nodes and the user of the network. Using a real-life metaphor, a sensor node can be abstracted as the “sensing cells” of a living system and the base station as the “brain” of such system, the ultimate destination of all signals generated by the cells. There is a difference between the metaphor and reality, though: It is possible to have more than one base station, or even use a mobile base station.

The “living system” metaphor can be further extended if the internal architecture of the network, or how the nodes group themselves to achieve their goals, is taken into account. In hierarchical architectures, the network is divided into clusters or groups of nodes, where organizational decisions are made by a single entity called “cluster head”. Such nodes can behave as the “spinal cord” of the system. On the other hand, it is also possible to combine the previous architecture with a flat architecture, where all the nodes contribute in the decision-making process. As a result, the system can be more robust against any internal failure of its elements. A representation of the structure of a sensor network as a “living system” can be seen in Figure 1.

2.2 Security Problems

Among all the open problems that sensor networks as a paradigm has to face, security is one of the most important [2]. The sensor nodes, the environment, or the communication channel can be manipulated by any malicious adversary for its own benefit. The first cause of this problem is the hardware constraints inherent to the nodes: due to their small size, their energy consumption requirements, and their limited computational capabilities, it is very difficult to incorporate the mechanisms used as a foundation for secure protocols. The second cause is the public nature of both the wireless channel and the sensor nodes: any device can listen to the communication flow, and the nodes can be accessed and tampered by any external entities. Finally, the third cause is the distributed nature of the sensor network: all protocols have to cooperate for pursuing a common goal, and any internal or external problem may hinder the provision of the network services.

Since sensor networks are very vulnerable against attacks, it is necessary to have certain mechanisms that can protect the network before, during, and after any kind of attack. One of the most important tools for ensuring the security of the network and its services are the security primitives. As mentioned in the introduction, we will consider that those primitives are Symmetric Key Encryption (SKE), Public Key Cryptography (PKC), and Hash functions. Hash and SKE primitives are the building blocks to offer a basic protection of the information flow, assuring the confidentiality and integrity of the channel. Moreover, PKC allow the authentication of the peers involved in any information exchange, thus protecting from the participation of external entities and eliminating the problem of a malicious insider trying to use more than one identity (sybil attack [3]).

These primitives are not sufficient by themselves for guaranteeing the overall security of the whole network, since any malicious insider located inside the network can disrupt its behavior regardless of the protection provided by those primitives. Nevertheless, they are essential for providing basic security to the core protocols of the network, that is to say, the minimal set of protocols required to provide services, such as routing, data aggregation and time synchronization. These core protocols provide packet transmission from one node to another node, grouping a set of sensor readings into one single piece of data, and synchronizing the clocks of the network, respectively.

Moreover, it is possible to create better network services based on the primitives. For example, if the authenticity of a code that is being uploaded to the node using the wireless channel can be assured, it will be possible to update the behavior of the node or to execute a mobile agent. Also, in most services, sensor nodes have to exchange certain information in order to obtain a global perspective of a situation from local information. Authenticating the source of such information and assuring its integrity can lead to the creation of better and more efficient trust management algorithms, intrusion detection systems, location procedures, and so on.

A final note regarding symmetric security primitives is the need of having a key management system (KMS) for constructing a secure key infrastructure. In most cases, it is not possible to know beforehand where the nodes are going to be located inside the network, but a single sensor node needs to know the keys it shares with its neighbors in order to open a secure channel. This is a well-researched topic, with many types of protocols that try offer the most adequate properties for a certain context [4].

3 Primitives Analysis and Requirements

Each of the aforementioned primitives, such as SKE, provides a different functionality and has different requirements in terms of processing power, word size, etc. Therefore, it is important to review the functionality of the primitives in order to analyze their theoretical suitability to the existent sensor node hardware. Although there are many algorithms that implements the requirements of the primitives, only a few of them can fit into the constrained environment of a sensor node (check [5] for a deeper review on the subject and the other algorithms). Is in these algorithms where this section will focus on.

3.1 Symmetric Key Encryption Primitives

There are basically two types of SKE primitives: block ciphers and stream ciphers. Block ciphers operates on fixed-length groups of bits, termed blocks, with an unvarying transformation. Most block ciphers combine multiple rounds of repeated operations, where the key applied in every round is obtained from a transformation of the original key using a key schedule. In all block ciphers, it is necessary to use a mode of operation, such as Counter (CTR), Cipher Block Chaining (CBC), and Counter with MAC (CCM) for encrypting messages longer than the block size. On the other hand, a stream cipher operates on individual digits one at a time, and the transformation varies during the encryption. Note that the distinction between the two types is not always clear-cut: a block cipher, when used in certain modes of operation like CTR, acts effectively as a stream cipher.

One of the simplest and fastest block cipher algorithms is Skipjack [6]. Skipjack is a 64-bit block cipher with an 80-bit key, that encrypts 4-word data blocks (16 bits each block) by alternating between two stepping rules, named A and B. Each rule can be described as a linear feedback shift register with additional non linear keyed G permutation. Its key schedule is also straightforward: just cyclically repeating the key enough times to fill the key schedule buffer. A downside of this algorithm is the small key length.

Another simple algorithm is RC5 [7], which is a block cipher with variable parameters: block size (64-bit suggested), key size (128-bit suggested) and number of rounds (20 rounds suggested). The encryption routine consists of three primitive operations over two $b/2$-bit registers (e.g. if b = 64-bit, register size = 32): integer addition, bitwise XOR, and variable rotation. Its key schedule is a bit more complex, though. Based on RC5, RC6 [8] is another block cipher that has two main new features with respect to RC5: the inclusion of integer multiplication and the use of four $b/4$-bit working registers instead of two $b/2$-bit registers (where b is suggested as 128).

Other complex block ciphers are AES [9], which is based on the Rijndael algorithm [10], and Twofish [11]. AES uses a fixed block size of 128 bits and a key size of 128, 192 or 256 bits. AES operates on a 4×4 array of bytes, termed the state. For encryption, each round of AES (except the last round) consists of four stages (ignoring the key schedule): AddRoundKey, where the subkey is combined with the state using the XOR operation, SubBytes, where each byte in the state is replaced with its entry in a fixed 8-bit lookup table, ShiftRows, where bytes in each row of the state are shifted cyclically to the left, and MixColumns, where each column of the state is multiplied with a fixed polynomial $c(x)$. Note that most of AES calculations are done in a special finite field.

Twofish is, like AES, a 128-bit block cipher with key sizes up to 256 bits. Twofish's distinctive features are the use of four different, bijective, key-dependent 8-by-8-bit S-boxes, and a relatively complex key schedule. Other building blocks are a 32-bit pseudo-Hadamard transform ($PHT(a, b) = \alpha, \beta$, where $\alpha = a + b \ mod \ 2^{32}$ and $\beta = a + 2b \ mod \ 2^{32}$), and a single 4-by-4 maximum distance separable (MDS) matrix over $GF(2^8)$.

Regarding stream ciphers, RC4 [12], a XOR-based stream cipher, is notorious for its simplicity. RC4 generates a pseudorandom stream of bits which, for encryption, is combined with the plaintext using XOR. It is extremely simple: its building blocks are bitwise ANDs, additions and swappings. Unlike the block size of other cryptographic algorithms, RC4 uses a 8-bit block size, and it does not require many memory resources. Note that there is an ongoing process as of 2007 (eSTREAM, the ECRYPT Stream Cipher Project [13]) that is aiming to identify and standardize new stream ciphers in hardware and software platforms that might become suitable for widespread adoption.

3.2 Hash Function Primitives

Usually, cryptographic hash functions process the input text in α-bit blocks to generate a β-bit hash value. Instances of cryptographic hash functions are the SHA-1 algorithm, with $\alpha = 512$ and $\beta = 160$, the SHA-2 algorithm with digest length of 256 (SHA-256), with $\alpha = 512$ and $\beta = 256$, and the RIPEMD-160 algorithm, with $\alpha = 512$ and $\beta = 160$. Hash functions are usually used to build other cryptographic primitives like Message Authentication Codes (MAC), which can protect both a message's integrity as well as its authenticity. Note that SKE primitives can also be used for constructing these MACs using the CBC mode of operation, in a process called CBC-MAC.

SHA-1 [14] is an extended, albeit slightly flawed, hash algorithm. It uses the following operations: XOR, AND, OR, NOT, additions, and rotations. Since the complexity required for finding a collision is 2^{63} [15], applications designers should use stronger hash functions from the same family such as SHA-256. On the other hand, RIPEMD-160 [16] is a hash algorithm with no known flaws that uses operations such as rotations, permutations, shifts, and others. Both SHA-1 and RIPEMD-160 use a word size of 32 bits for their internal operations. It is also important to note that, as of 2007, there is a research effort for creating a new hash function standard [17], which could be resilient against the known SHA attacks.

3.3 Public Key Cryptography Primitives

The most famous asymmetric algorithm is RSA [18], which was the first algorithm known to be suitable for signing as well as encrypting, and it is being widely used on many applications such as electronic commerce. Nonetheless, its operational requirements are known to be very expensive for resource-constrained microprocessors. As a result, in the context of sensor networks, it is necessary to use new algorithms based on more optimal approaches, such as Elliptic Curve Cryptography.

Elliptic Curve Cryptography (ECC) [19] is based on the algebraic structure of elliptic curves (i.e. a plane curve defined by an equation of the form $y^2 = x^3 + ax + b$) over finite fields $\mathbb{F}_p$ (of odd characteristic) or $\mathbb{F}_{2^m}$ (of characteristic two). The problem of finding a solution to an equation $a^b = c$ given a and c is known as the discrete logarithm problem, and it is the base of the security of these primitives. ECC includes many different types of primitives, such as ECDSA for signatures and ECDH for key agreement.

The main primitive operation in any ECC-based algorithm is the scalar point multiplication: A ECDSA signature is primarily one point multiplication, while a verification is mainly two point multiplications. At the same time, a point multiplication is achieved by repeated point addition and doubling, which requires an integer inversion calculated over affine coordinates. These operations are quite expensive in computational terms, although there are some known optimizations such as Shamir's trick for reducing the verification time and the use of projective coordinates to avoid the expensive inversion operations. Still, these point multiplications are simpler that the main primitive operation of RSA, exponentiations.

There are other approaches that can be useful for sensor nodes, such as NtruEncrypt [20], Rabin's scheme [21], and $\mathcal{MQ}$-schemes [22]. NtruEncrypt and its associated signature scheme NtruSign are relatively new asymmetric crypto algorithms based on arithmetics in a polynomial ring $R = \mathbb{Z}(x)/((x^N - 1), q)$. Their strength comes from the hardness of the Shortest Vector Problem (SVP) and the Closest Vector Problem (CVP) in lattices of high dimensions. Both mainly use as primitive operations simple polynomial multiplications for encryption and signing, thus they claim to be faster than other asymmetric encryption schemes. However, their security have not been as deeply analyzed as other asymmetric primitives.

On the other hand, Rabin's scheme is an old algorithm based on the factorization problem of large numbers, thus its security is similar to RSA. The main feature of this algorithm is its speed, though: the encryption operation consists on a simple squaring operation. Aside from the decryption speed, it has a peculiar disadvantage: each output of the Rabin function generates three false results in addition to the correct one. Therefore, extra complexity is required to identify which of the four is the true plaintext.

Finally, the multivariate public-key cryptosystems (or $\mathcal{MQ}$-schemes) are based on the hard problem of computing $w = V^{-1}(z) = (\omega_1, ..., \omega_n) \in K^n$ given a quadratic polynomial map $V = (\gamma_1, ..., \gamma_m) : K^n \rightarrow K^m$. In particular, the security of the variant known as the enTTS(20,28) algorithm, which belongs to the STS-UOV family of $\mathcal{MQ}$-signatures, is comparable to RSA-1024, using 160-bit hashes and 224-bit signatures. The weakness of this algorithm is the length of the keys: 8680 bytes for the public key and 879 bytes for the private key.

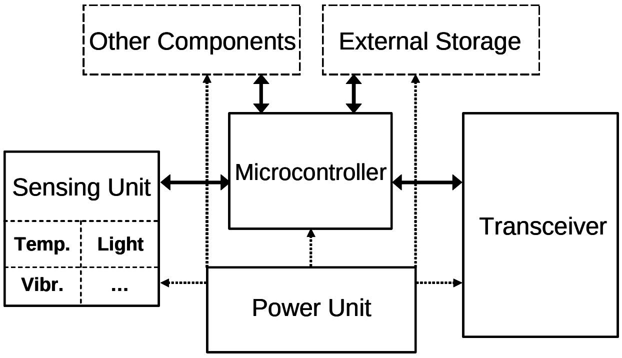

Figure 2: Overview of the Architecture of WSN

4 Sensor Hardware

A sensor node is made up of four basic components: sensing unit, processor unit, transceiver, and power unit [1], as seen in Figure 2. The sensing unit consists of an array of sensors that can measure the physical characteristics of its environment, like temperature, light, vibration, and others. The processing unit is, in most cases, a microcontroller, which can be considered as a highly constrained computer that contains the memory and interfaces required to create simple applications. The transceiver is able to send and receive messages through a wireless channel. Finally, the power unit provides the energy required by all components, and such energy may come from either a battery or from renewable sources. Note that there can be also other components depending on the needs of the application, like external data storage (e.g. flash memory), location devices (e.g. GPS chips), or cryptographic chips.

We will focus our study on the processing unit and the transceiver because they are the most important parts of the node from the point of view of implementing cryptographic primitives. The computational power and other resources located in the microcontroller will influence the capacity of the node to calculate the basic operations of a primitive, whereas the data throughput capabilities of the transceiver will influence over the capacity of negotiating secret keys and the overhead of protecting the packets. Regarding hardware assistance for executing cryptographic operations, at present there are no sensor nodes that provide this kind of service to the application developer, although some transceivers do offer symmetric primitives support in their hardware.

4.1 Microcontroller Devices

Microcontrollers are specially suitable for the wireless sensor network environment, due to their cost-effectiveness: a microcontroller used in a sensor node has enough computational capabilities and memory for executing simple tasks while consuming as less energy as possible. The selection of a microcontroller depends on what services has to provide to the node in terms of energy consumption, program and data memory, storage, speed, and external IO ports.

Note that a single manufacturer provides a whole array of microcontrollers to the market, but in most cases only one or two chips from the same manufacturer are used by sensor node companies (e.g. Atmel and the ATmega128L). As a side effect, nodes from different companies can share the same microcontroller (e.g. the BTNode and the Micaz), even if their overall architecture is completely different. Therefore, the impact of these nodes in the protocols and services that have to be implemented inside it, such as cryptographic algorithms, will be similar.

It is possible to classify the microcontrollers used in sensor nodes by their overall capabilities. Some of them are extremely constrained, and even unable to support the “de-facto” standard operative system for sensor nodes, TinyOS. These devices will be called “Weak”. An example of this type of device is the PIC12F675, which was used by the uPart node [30]. On the other side of the spectrum, there are microcontrollers that are as powerful as the microprocessors used in PDAs, and can host complex operative systems or Java-based virtual machines. These devices will be named “Heavy-Duty”. Examples of these devices are the PXA271 and the ARM920T, which are used in the iMote2 [23] and the SunSPOT [31], respectively.

Finally, the devices that are resource-constrained but powerful enough to hold a complex application will be known as “Normal”. This is the most common type of device for sensor nodes, and there are many microcontrollers that fall into this category. One of them is the ATmega128L, used in the Micaz [23], Mica2 [23], BTNode [24], and MeshBean [25]. Another example is the MSP430F1611, which is integrated in the Tmote Sky [26], telosb [23], EyesIFXv2.1 [27], and Tinynode 584 [28]. Other devices are the MSP430F14x, which is used by the ESB nodes like ScatterNode [29], and the PIC18F6720, used as the core of the zPart and pPart nodes [30].

	Weak	Normal				Heavy-Duty	
Model	PIC12F675	PIC18F6720	MSP430F14x	MSP430F16x	ATmega128L	PXA271	ARM920T
Frequency	4Mhz	20Mhz	4Mhz	8Mhz	8Mhz	13(416)Mhz	180Mhz
Word size	8bit	8bit	16bit	16bit	8bit	32bit	32bit
RAM memory	64B	4kB	2kB	10kB	4kB	256kB	512kB
Inst. memory	1.4kB	128kB	60kB	48kB	128kB	32MB	4MB
Power (awake)	2.5mA	2.2mA	1.5mA	2mA	8mA	31-44mA	40-100mA
Power (slept)	1nA	1μA	1.6μA	1.1μA	15μA	390μA	40μA

Table 1: Microcontrollers used in the sensor network market.

A summary of the different hardware microcontrollers and their capabilities (such as frequency, word size, RAM memory, Instruction memory, and so on) is shown in table 4.1. An important point about these microcontrollers is that there is no hardware support for any kind of primitive, although the PXA271 does supply the full MMX instruction set and the integer functions from SSE, and both could be used to optimize some implementations (cf. section 5.1).

It is clear that the “Weak” type of nodes are unable, at present, to execute any block cipher algorithm such as AES, RC5, Skipjack or ECC, because although it has enough memory size to hold a private key, there is no room for the primitives required to compute the necessary calculations (e.g. the code for ECC in a MSP430 is approximately 18.8KB [32], while the code required for using block ciphers is approximately 7KB) [33]. The same problem, lack of memory space, is shared by hash functions (e.g. the code of SHA1 in a MSP430 is approximately 2KB [34]) and stream ciphers (RC4 requires an internal state of 256 bytes in RAM). Therefore, encryption in these nodes should be performed or aided by HW modules.

The “Heavy-Duty” type of nodes are powerful enough to cope with any kind of primitives, either symmetric or asymmetric, via software. It is in the “Normal” type of nodes, like the Micaz and the TMote Sky (telosb), where the challenges lie. Thanks to the, albeit limited, memory capacity and computational power of these nodes, it is possible to create software implementations of the algorithms. In any case, these software implementations have to be compatible, that is, smaller enough to leave room for the operative system and for the core protocols and services offered by the node, and also suitable, that is, faster enough to be usable during the network lifetime.

4.2 Transceivers

One of the major features of sensor nodes are their ability to send and receive messages through a wireless channel. Thus, it is necessary to integrate a transceiver (i.e. transmitter-receiver) unit in the node. These transceivers have to offer an adequate balance between a low data rate (e.g. between 19.2Kbps and 250Kbps) and a small energy consumption in low-voltage environments (i.e. around 3V), allowing the node to live for a extended period of time. Because of these reasons, standards such as 802.11 cannot be used in this kind of networks. The same situation found on the microcontroller market can be found in this one: only one or two chip models from the same manufacturer are used by sensor node companies, and some sensors share the same transceiver hardware.

Transceivers in sensor networks can be divided into two categories: narrowband radios and wideband radios. Narrowband radios have less throughput and are more susceptible to noise, but they have less power consumption and faster wakeup times. These advantages and disadvantages of narrowband are reversed in wideband radios: they are faster and more robust, but also more power-demanding and slower to wake up. Narrowband radios work at lower frequencies (433Mhz and 868Mhz in Europe, 915Mhz in USA), while wideband radios usually work at higher frequencies (2.4Ghz). In these higher frequencies we find two non-compatible standards that fits into the wideband category: Zigbee and Bluetooth. However, nodes usually just use the 802.15.4 MAC layer below the Zigbee standard, implementing its own protocol stack.

The most common narrowband transceiver is the CC1000, used by the BTNode [24], Mica2 [23] and Mica2Dot [23]. A chip with similar characteristics, the CC1020, is used on the MSB [29] nodes. Other chips are the TR1001, used by the Scatternodes [29], and the XE1205, used by the TinyNode 584 [28]. On the wideband side, the most common chip is the Chipcon CC2420, which implements the 802.15.4 standard and is used by the Micaz [23], TMote Sky [26], SunSpot [31], zPart [30], MeshBean [30] and IMote2 [23]. Other bluetooth-enabled nodes, like the BTNode [24], use the ZV4002 chip. A summary of the capabilities of these hardware transceivers can be seen in table 4.2.

	Narrowband				Wideband	
Model	CC1000	CC1020	XE1205	TR1001	CC2420	ZV4002
Frequency	300-1000Mhz	402-940Mhz	300-1000Mhz	868Mhz	2.4Ghz	2.4Ghz
Max. Data Rate	76.8kbps	153.6kbps	152.3kbps	115.2kbps	250kbps	723.2kbps
Modulation	FSK/OOK	FSK/GFSK/OOK	CPFSK	OOK/ASK	DSSS-O-QPSK	FHSS-GFSK
Turn on time	< 2ms	< 2ms	< 2ms	< 0.5ms	< 1ms	—
Power (RX)	9.6mA	19.9mA	14mA	3.1mA	18.8mA	65mA
Power (TX)	16.5mA	20.5mA	33mA	12mA	17.4mA	65mA
Power (slept)	$< 1\mu$A	$< 1\mu$A	$< 1\mu$A	$< 1\mu$A	$< 1\mu$A	140μA
Security	none	none	none	none	AES-128	SC-128

Table 2: Transceivers used in the sensor network market.

Both narrowband and wideband radios have enough data throughput to allow a large number of key negotiations between neighbors, resulting on the possibility of having a fast setup of the network. Also, the overhead imposed by the primitives (usually a 10% [33]) leaves enough room for other important data. Nevertheless, regarding the data rate, it is important to note that the maximum data rate shown in the table is the maximum theoretical data throughput. About hardware assistance, transceivers based on the 802.15.4 standard (e.g. CC2420) and on the Bluetooth standard (e.g. ZV4002) provides support for symmetric cryptography, allowing microcontrollers to spend their modest resources in other important tasks such as asymmetric cryptography. However, from the current consumption values, it is clear that Bluetooth was not designed with sensor networks in mind.

4.3 Suitability of the Security and Privacy Primitives

A sensor node needs to incorporate a set of cryptographic algorithms in order to provide the necessary primitives required by any security service. Nevertheless, the suitability of an algorithm is strongly dependant on the complexity of its building blocks and its internal operations.

Over all the SKE algorithms, the stream cipher RC4 is the most optimal for sensor nodes, since its block size (8-bits) and its behavior (only XOR, additions, ANDs, and swappings) makes it suitable for highly constrained microcontrollers. Skipjack is also a good candidate, since it is very simple in both its operation and its key schedule, and perfectly fits the architecture of the MSP430 family - 16-bit words.

RC5 and RC6 are not as appropriate as RC4 and Skipjack due to their register size of 32 bits and their high number of rounds (20), but their building blocks are extremely simple, resulting on (possibly) a small code size. Finally, both AES and Twofish are also optimized to work in 32-bit processors, but their complexity is higher. Nevertheless, the number of rounds is smaller, and some operations can be done natively over 8-bit registers; hence they are supposed to be faster.

Regarding hash functions, all the algorithms previously presented are optimized for 32-bit microprocessors. Consequently, the 8-bit and 16-bit microcontrollers used in sensor nodes will have problems to efficiently implement such functions. Again, the building blocks of these algorithms are quite simple, so it is possible to make small (code-wise) software implementations that calculate the hash values fast enough. Nevertheless, it should be interesting to develop secure and lightweight hash functions specially created for these types of nodes.

All PKC primitives are extremely complex and fairly unsuitable to small general-purpose microcontrollers. Even though, every primitive has its own advantages. The key size of any ECC-based algorithm is substantially smaller than the others, e.g. a 160-bit key is equivalent to a 1024-bit RSA key. Algorithms based on polynomial rings (e.g. NTRUEncrypt) performs only polynomial multiplications; thus, they should achieve a good average speed in both encryption and decryption. On encryption and verification Rabin's speed is, simply, unbeatable. At last, signatures on $\mathcal{MQ}$-schemes are very fast, and if the public key could be stored entirely on RAM, verifications could also be significantly faster than others.

5 Hardware Efficiency for Cryptographic Primitives

In a sensor network environment, the most effective solution in terms of using the cryptographic primitives would be to have hardware support in the microcontrollers. In this way, the overhead of the processing units could be lower, and the nodes would be able to execute more complex tasks. Unluckily, the only hardware support available for application designers in sensor nodes are provided by the radios implementing the 802.15.4 standard and by the MMX extensions included in the PXA271 microcontroller. Then again, there is an active research community working on the creation of simple and energy-efficient cryptographic hardware designs.

5.1 Existing hardware

All transceiver chips that implement the 802.15.4 standard can offer a suite of AES-based cryptography operations for assuring the confidentiality and integrity of the network packets. The available suites are AES-CTR (Encryption in Counter Mode), AES-CBC-MAC (Message Authentication Code using AES in Cipher Block Chaining Mode), and AES-CCM (Counter with CBC-MAC). AES-CTR provides only confidentiality by behaving as a stream cipher, AES-CBC-MAC provides only authentication by using AES in CBC-MAC mode, and AES-CCM provides both confidentiality and authentication. In order to use these security suites, the application must create an ACL (access control list) entry consisting on a source address, a destination address, a suite, and the secret key of 128 bits to be used between the source and the destination. Once the CC2420 receives a packet with security enabled, it looks up for an entry in the ACL, and applies the security suite if a match is found. Note that, although there can be up to 255 ACL entries, the standard does not specify a minimum size. Moreover, the only mandatory suite that the chip must offer by default is AES-CCM with a 64-bit MAC.

Unfortunately, the 802.15.4 standard, and all the chips that implement it, is not exempt of flaws [35]. One of the security suites, AES-CTR, is deeply flawed, since it does not properly support replay detection and it is possible to launch Denial of Service (DoS) attacks sending a single forged packet. Also, the acknowledgement packets are not protected by a MAC, thus it is possible to forge them. Other minor problems include deleting the ACL when entering low power mode. As a result, chip manufacturers and application designers must take into account the inherent problems of the standard.

The most common 802.15.4 chip integrated in sensor nodes, the Chipcon CC2420, implements all the recommended security suites (including the flawed AES-CTR), and also provides an additional simple encryption mode where a 128 bit plaintext is encrypted with a 128 bit secret key. The hardware implementation of these suites is quite fast: the CC2420 is able to apply AES-CCM to a single packet in 222μs, and the simple encryption mode works in only 14μs. However, its ACL has only two entries, thus the chip only provides “out-of-the-box” support for two neighbors. Therefore, the application developer has to provide a workaround for taking advantage of the hardware capabilities. An example would be to implement CBC-MAC in software by using the simple encryption mode [36].

Regarding MMX, this SIMD instruction set was designed by Intel and introduced in 1997 as part of the Pentium MMX microprocessors. Nowadays, this instruction set is also part of the PXA27X family of microprocessors as “Wireless MMX”. This extension provides 16 data registers of 64 bits and 8 control registers of 32 bits, and is able to perform two 32-bit, four 16-bit, or eight 8-bit operations in a single instruction. These operations range from simple mathematical and logical operations like rotating, adding and multiplying to other specific operations such as calculating the vector maximum/minimum.

This level of parallelism can help to the software implementation of block ciphers such as AES or Twofish [37], which use simple operations such as XOR over a set of 8-bit registers. In the case of AES, it has a large internal parallelism, thus there is potential for optimization. On the other hand the internal operations of Twofish, such as the Pseudo-Hadamard Transformation, are not parallel-friendly, although it is still possible to optimize certain parts of the algorithm.

5.2 Research efforts

Most hardware extensions proposed by the research community for constrained devices such as sensor nodes aim to improve the execution of asymmetric cryptography primitives, but they only exist as a mere proof-of-concept. It is then desirable that these extensions are implemented as part of the microcontrollers or as external chips so that they can be used for leveraging the load of the computing core.

Nearly all efforts in the research community have been aimed on providing hardware support to Elliptic Curve Cryptography (ECC). Wolkerstorfer et. al. [38], as well as Kumar et. al. [39] developed integrated chips that can provide an excellent performance for signing a message using ECDSA, but at the cost of a high operating frequency. In the case of Wolkerstorfer, the chip has an area of 23000 gates implemented in 0.35μm CMOS technology, operates at 68.5Mhz, and is able to execute a one point multiplication over $\mathbb{F}_{2^{191}}$ in only 6.67ms. On the other hand, the chip created by Kumar and Paar is slower, providing a point multiplication over $\mathbb{F}_{2^{131}}$ in 18ms, but has less gates (almost 12k), and works at a slower operating frequency (around 13Mhz). These implementations are not applicable to most sensor nodes, although some microcontrollers, like the PIC18F6720, PXA271, and the ARM920T, would be able to support them.

The implementation by Gaubatz et. al. [40] is more oriented to highly constrained sensor nodes, since its operational frequency is only 500kHz. In their ECC scalar point multiplication architecture, operations are performed over $\mathbb{F}_{p_{100}}$, occupying a chip area equivalent to 18720 gates in 0.13μm CMOS technology, and consuming just under 400μW in signing and encrypting messages. ECDSA signatures and ECMV decryptions are performed at around 410ms, and ECDSA verifications and ECMV encryptions are performed at around 820ms. As for the optimizations, they efficiently implement modular reduction, achieve a fast inversion algorithm, and implement all arithmetic primitives in a bitserial fashion.

This implementation was further improved by Batina et. al. [41]. Their Elliptic Curve Processor (ECP) included a Modular Arithmetic Logic Unit (MALU) capable of computing modular additions and multiplications, using the same cells for both purposes without having a full-length array of multiplexors. Moreover, squaring is considered a special case of multiplication, and inversion is avoided mostly by use of projective coordinates. The results are promising: using 8104 gates (12kgates including RAM) in 0.13μm CMOS technology, one point multiplication over $\mathbb{F}_{2^{131}}$ is performed in only 115ms, consuming less than 30μW in a operational frequency of 500kHz.

ECC is not the only primitive that researchers have considered for implementing PKC in sensor networks. In [40], Gaubatz et. al. proposed the NtruEncrypt public key cryptosystem as a possible candidate. This solution employs a small number of gates (3000) and only consumes about 20μW in an operational frequency of 500kHz. Encryption and verification operations are performed at around 58ms, decryptions are calculated in almost 117ms, and messages are signed in almost 234ms. Another scheme, Rabin's scheme, was proposed by the same authors [40]. The design used less than 17000 gates with an average power consumption of 148.18μW, and achieved speeds as lower as 2.88ms for encrypting and verifying messages. However, the downside comes from the decryption and signature operations: the processor has to employ 1.089s for executing them.

Finally, the Multivariate public key cryptosystem was also considered due to its benefits regarding Quantum Computing. The implementation made by Yang et. al. [42] only included signature generation, since it was mostly oriented to RFID tags. The optimal form, the circulant (LPSQR) form of enTTS(20,28), uses 17000 gates in 0.25μm CMOS technology, and has a signature time of only 44ms and a energy consumption of 25μA in an operational frequency of 100kHz. This speed is achieved thanks to the substitution of the Lanczos' method used in the internal operations with the symmetric Gaussian elimination.

	ECC				NTRU	Rabin	$\mathcal{MQ}$
	Wolkerstorfer	Kumar & Paar	Gaubatz	Batina	Gaubatz	Gaubatz	Yang
Gates	23000	12000	18720	12000	3000	17000	17000
Technology	0.35μm	0.35μm	0.13μm	0.13μm	—	—	0.25μm
Frequency	68.5MHz	13.5Mhz	500khz	500kHz	500kHz	500kHz	100kHz
Field	$\mathbb{F}_{2^{191}}$	$\mathbb{F}_{2^{131}}$	$\mathbb{F}_{p_{100}}$	$\mathbb{F}_{2^{131}}$	—	—	—
Point Mult.	9.98ms	18ms	$\sim$ 400ms	115ms	—	—	—
Encryption	—	—	—	—	58ms	2.88ms	—
Decryption	—	—	—	—	117ms	1.089s	—
Signing	—	—	—	—	234ms	1.089s	44ms
Verifying	—	—	—	—	58ms	2.88ms	—
Energy	n.a.	n.a.	$\sim$ 133μA	$\sim$ 10μA	$\sim$ 6.6μA	$\sim$ 49.4μA	25μA

Table 3: Summary of Hardware prototypes for PKC.

Table 5.2 shows the different hardware prototypes categorized by their underlying primitive, considering the operating voltage at 3V. Note that the encryption time and other time-related rows are empty for ECC-based prototypes because the time of completing a scalar point multiplication largely influences over the operations of the algorithms (cf section 3.3). From the table, it is possible to notice that Batina et.al. hardware implementation is the best ECC prototype. Comparing it with other primitives, the prototype consumes slightly less energy, but the time for calculating an internal operation (a point multiplication for a ECC-based primitive) is a bit larger. The $\mathcal{MQ}$ prototype is really powerful for signing, but the size of its key is too large. Regarding NTRU, it is well-balanced, but its signature is 1.169 bits, thus it needs at least 5 packets in order to send a message. Lastly, Rabin is extremely fast on encryption and verifying, although its signatures requires 512 bits.

6 SW Implementations

6.1 Symmetric Key Encryption and Hash Functions

Although the microcontrollers used as the computing core of sensor nodes are highly constrained, it is possible to execute various instances of SKE and Hash functions only by software means. The main problem that these software implementations face is achieving an encryption speed smaller than the "byte time", i.e. the theoretical duration that it takes to transmit a single byte of data over the radio. If the encryption process is slower than the byte time, there can be problems in high throughput scenarios such as the radio waiting for the processor to complete the cryptographic operation. For a speed of 19.2kbps, e.g. the CC1000, the byte time is approximately 420μs. For a speed of 250kbps, e.g. the CC2420, the byte time is approximately 32μs.

In 2003, Ganesan et. al. [43] analyzed the encryption overhead of SKE and Hash function primitives in highly constrained microcontrollers, such as the ATmega128. The results were highly promising: the number of seconds required to encrypt a single byte of plaintext were 26μs for RC5, 21μs for IDEA, and 6μs for RC4. However, hash functions did not perform as well as the SKE: the SHA-1 algorithm took 7777μs for digesting 64 bytes. Nevertheless, the average code size of a single primitive was quite good: less than 4000 bytes of ROM.

Symmetric key encryption primitives were finally available to sensor nodes as of 2004 through the TinySec package [33]. This package implemented a nesC interface to access to the cryptographic operations: Skipjack (48μs per encrypted byte, unoptimized) and RC5 (33μs per encrypted byte, optimized). Also, the package was able to provide link layer security (confidentiality, integrity and authentication) to the information flow by using these primitives. This particular implementation did not need any hash function for the integrity process since the message authentication code was calculated using CBC-MAC. However, TinySec only works with Mica and Mica2 nodes over TinyOS 1.x, thus it does not provide security to other CC2420-based nodes such as the Micaz or the TMote Sky.

Other instances of software-based SKE primitives were analyzed by Wei Law et. al. [44] and Jun Chio and Song [45]. SKE algorithms such as Twofish and Rijndael (AES) had an average encryption time of 50μs per byte, while an optimized Skipjack was able to provide an encryption time of 25μs per byte. Skipjack was also the best algorithm in terms of memory requirements, with approximately 2600 bytes of ROM, whereas the other algorithms needed an average ROM size of 8000 bytes. Nevertheless, the stream cipher RC4 was undoubtly the most optimal algorithm in terms of memory overhead: its memory footprint is only 428 bytes.

6.2 Public key Cryptography

The use of PKC in sensor nodes using software implementations was usually rejected as “not possible” in a sensor network environment until 2004, where some studies [46] claimed that Elliptic Curve Cryptography (ECC) seemed to be a good candidate for implementing PKC over sensor networks due to its small key sizes, faster computation, as well as memory and energy savings. This claim was finally proven by Malan et.al. [47] with the introduction of the EccM 2.0 library. This library implemented ECC primitives over $\mathbb{F}_{2^p}$ with a 163-bit key, spending an average running time of approximately 34 seconds for point multiplication and shared secret key generation.

Although these results confirmed the possibility of using PKC in sensor nodes, its running time was still too high for utilizing it outside research testbeds. At that moment, it was unclear if the ECC calculations could be further optimized. However, as pointed out by Gura et. al. [46], the ECC module could have a better performance if implemented over $\mathbb{F}_p$ rather than over $\mathbb{F}_{2^p}$. This optimization was grounded on the use of projective coordinates rather than affine coordinates, in order to avoid expensive inversions - an approach also taken by Batina et. al. [41] in their hardware implementation (cf. Section 5.2). This and other optimizations where implemented by Liu and Ning (TinyECC [34]) and Wang and Li (WMECC [32]) on TinyOS using the NesC language, obtaining excellent results with, in most cases, a running time smaller than 2 seconds for signing and verifying.

TinyECC and WMECC have some optimization techniques in common, but they are not completely equal in their design and have subtle differences. The use of projective coordinates is mandatory in both implementations, but TinyECC uses the weighted projective (Jacobian) representation (X, Y, Z) and WMECC use a mixed representation with modified Jacobian coordinates (X, Y, Z, aZ^4) and Affine coordinates (X, Y). Also, both implementations make use of the sliding window method (i.e. grouping a scalar k into s-bit clusters) for improving the performance of scalar point multiplications by pre-computing the point multiplication, and use the Shamir’s trick for reducing the signature verification time by doing multiple point multiplications simultaneously. However, WMECC use a small window value for both the sliding window method and Shamir's trick ($s = 4$ and $w = 1$, respectively), thus it does not require the expensive precomputation phase that TinyECC must perform for initializing the ECDSA system.

An identical optimization that both implementations include is the consideration of the underlying field primes p as pseudo-Mersenne primes (i.e. $p = 2^n - c$) to allow optimized modular reduction, which provides a better performance for the modular multiplications and the modular square. On the other hand, WMECC only implements the secp160r1 elliptic curve domain, while TinyECC implements the sec128r1, secp128r2, secp160k1, secp160r1, secp160r2, secp192k1, and secp192r1 elliptic curve domains recommended by standards for Efficient Cryptography Group (SECG [48]). Since TinyECC has to support multiple key sizes, it includes a specific optimization, called Hybrid Multiplication, that can perform modular multiplication of large integers minimizing the number of extra registers.

	TinyECC		WMECC		TinyWMECC	
	Micaz	Telosb	Micaz	Telosb	Micaz	Telosb
Test - ROM size	28266	26048	57982	46156	29734	25774
Test - RAM size	2306	2327	1685	1657	1643	1599
ECC init	1.837s	-	1.809s	1.744s	1.809s	1.744s
ECDSA init	3.550s	5.225s	0s	0s	0s	0s
Pub. Key Gen.	1.788s	-	1.261s	1.425s	1.261s	1.425s
Signature	1.916s	4.361s	1.348s	1.498s	1.348s	1.498s
Verification	2.431s	5.448s	2.017s	2.207	2.019s	2.209s

Table 4: Software implementations of ECC

The results of an experiment conducted with the purpose of analyzing the overhead of both implementations in the elliptic curve domain secp160r1 are presented in Table 6.2. The "Test - ROM/RAM size" rows show the ROM and RAM used by the primitive and the test program, and the other rows show the time spent in executing the various steps of the primitive. These values were obtained after 20 rounds of execution in both Micaz and TMote Sky nodes. Note that the values from TinyECC in Telosb are obtained from the original source, since the experiment could not be reproduced in our laboratory. It can be seen that in both implementations the Micaz is slightly faster than the TMote Sky, mainly due to the extra cycles spent in unsigned multiplications on the MSP430 microcontroller.

Comparing both implementations, the WMECC package yield a better performance than the TinyECC package: It has no initialization time for the ECDSA, and the signature and verification times are slightly faster. However, its memory footprint is simply prohibitive, and it would be impossible to develop any kind of application for a TMote Sky. Fortunately, thanks to the component capabilities of the TinyOS operating system, it is possible to replace the resource-consuming SHA-1 component of WMECC with the more optimized version used in TinyECC. This results on a new version of the WMECC package (deemed TinyWMECC in table 6.2) that is significantly better. Using this new implementation, it can be perfectly possible to create PKC-based applications on a constrained node such as the TMote Sky.

7 Conclusions

Wireless sensor networks need cryptographic primitives. At this moment, it is clear that, although constrained, the existent hardware is able to afford software algorithms that implement these primitives. There exists some exceptions, like the PIC12F675 microcontroller, where sensor node designers should take into account that a node containing this type of highly constrained processing core requires a cryptographic chip in order to provide a minimal protection of the information flow.

For high throughput applications, the overhead of some SKE algorithms such as Twofish or AES may limit the total bandwidth of the network. But most sensor network applications do not use all the available bandwidth, since they have to waste as less energy as possible in order to monitor a certain environment for a long period of time. This kind of optimization reduces drastically the bandwidth of the network, and makes the extra overhead of protecting a packet using this primitives negligible. The extra operating cost of a secure application will be, then, minimal.

Public Key Cryptography is also a reality in sensor nodes. It is possible to use ECDSA algorithms for signing and verifying a message in the most common hardware platform used nowadays: the Micaz and the telosb (TMote sky). A problem of the existent implementations for the telosb was their code size (too big for the 48kB limit of the MSP430F16x) and its speed (too slow compared with the Micaz). Nevertheless, the component nature of TinyOS has allowed us to improve one of the existent packages, combining the performance and the low memory consumption of both implementations.

Nevertheless, in future sensor nodes designs, hardware chips can be used for saving ROM and RAM space on the microcontroller, thus making possible the execution of more complex applications. For example, in the field of asymmetric cryptography, it has been shown that very simple designs (less than 15000 gates) are able to efficiently provide ciphering and signing services for any kind of sensor node hardware. For certain application-driven sensor nodes, it can be also possible to use other asymmetric primitives instead of ECC. For example, nodes with a high RAM and with the sole purpose of sending authenticated messages to the base station may use $\mathcal{MQ}$-schemes such as enTTS(20,28) due to its fast signing algorithm. It is expected that in the future new and optimized hardware designs will appear.

As final thoughts, the implementation of existing standards, such as the AES algorithm in the 802.15.4-enabled chips, should be able to provide the maximum level of security. Failing to do so may result in suboptimal products in terms of security, such as the CC2420 and its ACL management. Also, it should be noted that most hash functions are optimized to work with 32-bit microprocessors, and the average bit size of a sensor node microcontroller is 8-bit and 16-bit. For that reason, it would be interesting to develop new hash functions specially designed with these constrained environments in mind. Finally, at present there is no implementation that can provide symmetric key encryption to sensor nodes other than mica2, although there are some efforts like AMSecure [49] that try to do so taking advantage of the 802.15.4 hardware resources.

Acknowledgments

This work has been partially supported by the SMEPP project (EU-FP6-IST 0333563) and the CRISIS project (TIN2006-09242). The first author has been funded by the Ministry of Education and Science of Spain under the “Programa Nacional de Formacion de Profesorado Universitario”.

References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci (2002). *Wireless sensor networks: a survey.* Computer Networks: The International Journal of Computer and Telecommunications Networking, vol. 38, no. 4, pp. 393-422, March 2002.

[2] J. P. Walters, Z. Liang, W. Shi, and V. Chaudhary. *Wireless Sensor Network Security: A Survey.* Security in Distributed, Grid, and Pervasive Computing, Editor: Yang Xiao, Auerbach Publications, CRC Press, ISBN 0-849-37921-0, 2006.

[3] J. Newsome, E. Shi, D. Song, A Perrig. *The Sybil Attack in Sensor Networks: Analysis & Defenses.* IEEE 3nd International Workshop on Information Processing in Sensor Networks (IPSN'04), April 2004.

[4] C. Alcaraz, R. Roman. *Applying Key Infrastructures for Sensor Networks in CIP/CIIP Scenarios.* Proceedings of the 1st International Workshop on Critical Information Infrastructures Security (CRITIS 2006), Samos (Greece), August-September 2006.

[5] J.-P. Kaps, G. Gaubatz, B. Sunar. *Cryptography on a Speck of Dust.* IEEE Computer, vol. 40, no. 2, pp. 38-44, February 2007.

[6] NIST-CSRC. *SKIPJACK and KEA Algorithm Specifications, version 2.* 29 May 1998, http://csrc.nist.gov/CryptoToolkit/

[7] R. L. Rivest. *The RC5 Encryption Algorithm.* Proceedings of the 2nd International Workshop on Fast Software Encryption (FSE 1994), Leuven (Belgium), December 1994.

[8] R. L. Rivest, M. J. B. Robshaw, R. Sidney, Y. L. Yin. *The RC6 Block Cipher, v1.1.* August 1998, http://theory.lcs.mit.edu/ rivest/

[9] FIPS 197, *Advanced Encryption Standard (AES)*, November 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[10] J. Daemen, V. Rijmen. *The Design of Rijndael.* Springer, ISBN 3-540-42580-2, 2002.

[11] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, N. Ferguson. *The Twofish Encryption Algorithm: A 128-Bit Block Cipher.* Wiley, ISBN 0-471-35381-7, 1999.

[12] B. Schneier. *Applied Cryptography, 2nd edition.* Wiley, ISBN 0-471-12845-7, 1996.

[13] ECRYPT Network of Excellence. *eSTREAM, the ECRYPT Stream Cipher Project.* http://www.ecrypt.eu.org/stream/

[14] D. Eastlake, P. Jones. *US Secure Hash Algorithm 1 (SHA1).* RFC 3174.

[15] X. Wang, A. Yao, F. Yao. *New Collision search for SHA-1.* Rump Session of the 25th Annual International Cryptology Conference (CRYPTO 2005), Santa Barbara (USA), August 2005.

[16] H. Dobbertin, A. Bosselaers, B. Preneel. *RIPEMD-160, a strengthened version of RIPEMD.* Proceedings of the 3rd International Workshop on Fast Software Encryption (FSE 1996), Cambridge (UK), February 1996.

[17] NIST. *Plan for New Cryptographic Hash Functions.* http://www.nist.gov/hash-function

[18] R. Rivest, A. Shamir, L. Adleman. *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.* Communications of the ACM, vol. 21, no. 2, pp. 120126, 1978.

[19] I. Blake, G. Seroussi, N. P. Smart. *Elliptic Curves in Cryptography.* Cambridge University Press, ISBN 0-521-65374-6, 2000.

[20] J. Hoffstein, J. Pipher, J. H. Silverman. *NTRU: a Ring based Public Key Cryptosystem.* In proceedings of the 3rd Algorithmic Number Theory Symposium (ANTS 1998), Portland (USA), June 1998.

[21] M. O. Rabin. *Digitalized Signatures and Public Key Functions as Intractable as Factorization.* Technical Report MIT/LCS/TR-212, Massachusetts Institute of Technology (1979).

[22] C. Wolf, B. Preneel. *Taxonomy of Public Key Schemes Based on the Problem of Multivariate Quadratic Equations.* Cryptology ePrint Archive, Report 2005/077.

[23] Crossbow Technology, Inc. MicaZ Datasheet.
http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf

[24] Art of Technology AG. http://www.btnode.ethz.ch

[25] Meshnetics. http://www.meshnetics.com

[26] Moteiv Corporation. TMote Sky Datasheet.
http://moteiv.com/products/docs/tmote-sky-datasheet.pdf

[27] Infineon Technologies AG. http://www.infineon.com

[28] Shockfish SA. TinyNode Fact Sheet.
http://www.tinynode.com/uploads/media/SH-TN584-103.pdf

[29] Scatterweb GmbH. Scatternode Datasheet.
http://www.scatterweb.com/content/downloads/datasheets/ScatterNode(v1.0)-datsheet-doc1.1-en.pdf

[30] Smart-Its project. uPart Datasheet.
http://particle.teco.edu/upart/datasheets/upart014xilmt.pdf

[31] Sun Microsystems, Inc. Sun Small Programmable Object Technology.
http://www.sunspotworld.com/products/

[32] H. Wang, Q. Li. *Efficient Implementation of Public Key Cryptosystems on MICAz and TelosB Motes.* Technical Report WM-CS-2006-07, College of William & Mary, October 2006.

[33] C. Karlof, N. Sastry, D. Wagner. *TinySec: a link layer security architecture for wireless sensor networks.* Proceedings of 2nd International Conference on Embedded Networked Sensor Systems (SenSys 2004), Baltimore (USA), November 2004.

[34] A. Liu, P. Kampanakis, P. Ning. *TinyECC: Elliptic Curve Cryptography for Sensor Networks (Version 0.3).* http://discovery.csc.ncsu.edu/software/TinyECC/, February 2007.

[35] N. Sastry, D. Wagner. *Security considerations for IEEE 802.15.4 networks.* In Proceedings of 2004 ACM Workshop on Wireless security (Wise 2004), Philadelphia (USA), October 2004.

[36] K. Sun, P. Ning, C. Wang, A. Liu, Y. Zhou. *TinySeRSync: Secure and Resilient Time Synchronization in Wireless Sensor Networks.* In Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS'06), Alexandria (USA), November 2006.

[37] K.Aoki and H. Lipmaa. *Fast implementations of the AES candidates.* Proceedings of 3rd AES conference, New York (USA), April 2000.

[38] J. Wolkerstorfer. *Scaling ECC Hardware to a Minimum.* In ECRYPT workshop - Cryptographic Advances in Secure Hardware - CRASH 2005. Leuven (Belgium), September 2005. Invited Talk.

[39] S. Kumar, C. Paar. *Are standards compliant elliptic curve cryptosystems feasible on RFID?.* In Proceedings of Workshop on RFID Security, Graz (Austria), July 2006.

[40] G. Gaubatz, J.-P. Kaps, E. Öztürk, B. Sunar. *State of the Art in Ultra-Low Power Public Key Cryptography for Wireless Sensor Networks.* In Proceedings of the 2nd IEEE International Workshop on Pervasive Computing and Communication Security (PerSec 2005), Hawaii (USA), March 2005.

[41] L. Batina, N. Mentens, K. Sakiyama, B. Preneel, I. Verbauwhede. *Low-Cost Elliptic Curve Cryptography for Wireless Sensor Networks.* In Proceedings of the 3rd European Workshop on Security and Privacy in Ad hoc and Sensor Networks (ESAS 2006), Hamburg (Germany), September 2006.

[42] B.-Y. Yang, C.-M. Cheng, B.-R. Chen, J.-M. Chen. *Implementing Minimized Multivariate Public-Key Cryptosystems on Low-Resource Embedded Systems.* In Proceedings of the 3rd International Conference on Security in Pervasive Computing (SPC 2006), York (UK), April 2006.

[43] P. Ganesan, R. Venugopalan, P. Peddabachagari, A. Dean, F. Mueller, M. Sichitiu. *Analyzing and Modeling Encryption Overhead for Sensor Network Nodes.* In Proceedings of the 2nd ACM International Conference on Wireless Sensor Networks and Applications (WSNA 2003), San Diego (USA), September 2003.

[44] Y. W. Law, J. Doumen, P. Hartel. *Survey and Benchmark of Block Ciphers for Wireless Sensor Networks.* ACM Transactions on Sensor Networks, vol. 2, no. 1, pp 65-93, February 2006.

[45] K. Jun Choi, J.-I. Song. *Investigation of Feasible Cryptographic Algorithms for Wireless Sensor Network.* Proceedings of the 8th International Conference on Advanced Communication Technology (ICACT 2006). Phoenix Park (Korea), February 2006.

[46] N. Gura, A. Patel, A. Wander. *Comparing elliptic curve cryptography and RSA on 8-bit CPUs.* In Proceedings of the 2004 Workshop on Cryptographic Hardware and Embedded Systems (CHES 2004), Cambridge (USA), August 2004.

[47] D. J. Malan, M. Welsh, M. D. Smith. *A Public-key Infrastructure for Key Distribution in TinyOS Based on Elliptic Curve Cryptography.* In Proceedings of 1st IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON 2004), Santa Clara (USA), October 2004.

[48] SECG - Standards for Efficient Cryptography Group. http://www.secg.org/

[49] A. D. Wood, J. A. Stankovic. *Poster Abstract: AMSecure - Secure Link-Layer Communication in TinyOS for IEEE 802.15.4-based Wireless Sensor Networks.* Presented at the 4th ACM Conference on Embedded Networked Sensor Systems (SenSys 2006), Boulder (USA), November 2006.