On the Hardware Implementation Efficiency of Cryptographic Primitives

RODRIGO ROMAN [a], CRISTINA ALCARAZ [a], NICOLAS SKLAVOS [b]

[a] *University of Malaga, Spain, emails: {roman, alcaraz}@lcc.uma.es*
[b] *University of Patras, Greece, email: nsklavos@ieee.org*

Abstract. Security has been proven a crucial factor in the provision of data services and especially in the computer-related environments. While wired and wireless networks come to all sectors of everyday life, security tries to satisfy the growing needs for confidentiality, integrity and non-repudiation. There are many instances of security primitives and each one of them has different requirements in terms of processing power, memory, energy consumption, etc. Therefore, it is important to review the functionality of the less resource-demanding encryption algorithms in order to analyze their theoretical suitability to the existent sensor node hardware. Still, the constraints inherent to the sensor nodes advise against the total dependence on software-based implementations, even more in the case of expensive primitives.

Keywords. Security, Cryptographic Primitives, Hardware Implementations.

1. Introduction

While the wireless devices are coming to the offices and houses, the need for strong secure transport protocols seems to be one of the most important issues in the communications standards. From email services to cellular provided applications, from secure internet possibilities to banking operations, cryptography is an essential part of the today's users needs.

Recent and future computer networks have special needs for cryptography. They must support the three basic types of cryptography: Bulk Encryption, Message Authentication and Data Integrity. Most of the widely used wireless systems support all the above different types of encryption. Additionally, some systems offer to the users the choice to select among two or three alternative ciphers for each encryption operation. The user can select the best-suited algorithm according to the application needs. In most of the cases, the same encryption system implementation supports all the three different types of cryptography.

The standards for network applications and services are maturing and new specifications in security systems are being defined. This leads to a large set of possible technologies that a service provider can choose. Although organizations and forums seem to agree to the increasing need for secure systems with wide strength, security is a black hole in the computer networks because of the implementation difficulty. The security layers of many communication protocols use outdated encryption algorithms, which have been proved unsuitable for hardware implementations, especially for constrained implementation environments.

In general, encryption algorithms use complicated arithmetic and algebraic modifications, which are not appropriate all the time for integrations with high quality implementation performance. That’s why the integrations of the encryption algorithms allocate many of the system resources, in hardware or in software terms, in order to be implemented as separated components.

In many cases software applications have been developed, in order to support the security and cryptography needs. But, software solutions are not acceptable for the cases of computer-related environments, with high speed and performance specifications.

2. Security & Privacy in Hardware

In any environment, either physical or logical, there exists the need of maintaining someone or something safe, away from harm. This is the role of security. On any computer-related environment, security can be considered as a non-functional requirement that maintains the overall system usable and reliable, protecting the information and information systems. In order to comply with this non-functional requirement, such environments must assure the existence of certain properties by implementing new protocols or extending the current ones. These properties and protocols can be equally useful for another non-functional requirement, privacy, which is related to the capacity of controlling the flow of information about a certain subject or entities.

Major security properties related to information assurance are confidentiality, i.e. ensure that the information is accessed only by those who are inside the system, integrity, i.e. guarantee that the information has not been manipulated, authentication, i.e. certify that the source of the information is who claims to be, and non-repudiation, i.e. assure that no party can deny having participated in a part or the whole transaction. Other security properties focus more on the functionality of the overall system: Robustness and Availability are related to the capacity of the system of providing an acceptable level of service in the presence of problematic situations, whereas Resilience is the ability to withstand “Node Compromise” attacks that can try to take control of the whole system.

In a sensor network environment the existence of security, that is, the need of ensuring that the existent devices and protocols comply with the major security properties is extremely important due to the associated constraints to the elements of the network and their particular behaviour. The elements of a sensor network, the sensor nodes, are highly constrained in terms of computational capabilities, memory, communication bandwidth, and battery power. Additionally, it is easy to physically access these nodes because they must be located near the physical source of the events, and they usually are not tamper-resistant due to cost constraints. Furthermore, any device can access the information exchange because the communication channel is public.

As a result, any malicious adversary can manipulate the sensor nodes, the environment, or the communication channel for its own benefit. For these reasons, it is necessary to provide the sensor network protocols with basic security mechanisms that can guarantee a minimal protection to the services and the information flow. As a result, there is a need to protect the communication channels between the nodes, and to defend the core protocols of the network, i.e. routing, data aggregation, and time synchronization, and other non-essential protocols, such as node location, code update procedures, and other collaborative processes, against any external or internal influence.

The foundation of all these secure protocols, and the tools that can aid these systems on integrating the security properties into their operations, are the cryptographic primitives: Symmetric Key Cryptography (SKC), Public Key Cryptography (PKC), and Hash functions. These primitives alone are not enough to protect a system, since they just provide the confidentiality, integrity, authentication, and non-repudiation properties. Nevertheless, without these primitives, it would be nearly impossible to create secure and functional protocols.

There are many instances of these primitives (i.e. algorithms), and each one of them has different requirements in terms of processing power, memory and energy consumption, etc. Therefore, it is important to review the functionality of the less resource-demanding algorithms in order to analyze their theoretical suitability to the existent sensor node hardware. Still, the constraints inherent to the sensor nodes advise against the total dependence on software-based implementations, even more in the case of expensive primitives such as PKC (cf. Section 3). The time a single node spends executing these primitives could be used for other primordial tasks, or even for implementing the necessary steps that are required to provide other security properties. Consequently, it is crucial to analyse the influence of the implementation of these primitives on hardware over the behaviour of the node.

2.1. Symmetric Key Cryptography

Symmetric Key Cryptography (SKC) primitives are able to offer the necessary mechanisms for protecting the confidentiality of the information flow in a communication channel. For achieving this confidentiality level it is necessary that both the origin and destination share the same security credential (i.e. secret key), which is utilized for both encryption and decryption. As a result, any third-party that does not have such secret key cannot access the information exchange, thus the security properties of these primitives mainly resides on the complexity of their internal operations and the length of the keys.

There are two types of SKC primitives: Stream Ciphers and Block Ciphers. Stream Ciphers are simpler and faster, while Block Ciphers are more flexible and powerful. These two types are discussed more profoundly in the following sections.

2.1.1. Stream Ciphers

A stream cipher is a method of symmetric cryptography that takes as an input a flow of bits of variable size and transforms it into a ciphertext. For that purpose, these cryptographic algorithms combines, mainly by using simple operators such as XOR, those input bits with a pseudorandom key flow obtained from a cryptographic key and an initialization vector (IV). It is essential to use such IV in stream ciphers for avoiding situations where one plaintext produces the same ciphertext when encrypted with the same key.

One of the better known stream cipher algorithm, which is used by several protocols like Secure Sockets Layer (SSL) or Wi-Fi Protected Access (WPA), is RC4 [RC4_1996]. This simple algorithm was designed by Ron Rivest in 1987, and it is also known as ARC4 or ARCFOUR. Its simplicity is due to two main reasons. Firstly, it minimizes the execution time by using simple operations as XOR, AND, addition and shifting. And secondly, it uses a block size of 8-bit, consuming less memory space than others cryptosystems. Therefore, RC4 is highly suitable for those architectures with highly-constrained microcontrollers. However, RC4 must be implemented carefully for avoiding any attacks as the detected in WEP [Fluh2001], where the primitive was not properly initialized.

2.1.2. Block Ciphers

A symmetric block cipher operates on a group of bits with a fixed-size (usually 64 or 128 bits) known as blocks. A block of plaintext is transformed by using several rounds of mathematical operations into a block of ciphertext, which has the same size as the input. Besides of those operations, in every round most block ciphers also have to transform the original key, using a usually complicated process named key schedule. In cases where more than one block has to be encrypted (e.g. a plaintext that is larger than the block size), it is necessary to use a mode of operation. Moreover, if the plaintext is not a multiple of the block size, some modes of operation require padding that plaintext.

Operation modes are just a set of operations using the block cipher inputs and outputs. Some of them provide confidentiality, and a few of them provide authentication. Concretely, the earliest modes, as ECB (Electronic Codebook), CBC (Cipher Block Chaining), OFB (Output Feedback) or CBC (Cipher Block Chaining), provide mainly only confidentiality. However, operation modes such as CCM (Counter with MAC), GCM (Galois Counter Mode), OCB (Offset Codebook Mode) and others modes guarantee both security properties by including a Message Authentication Code (MAC). In addition, in order to generate some randomization in the process, these modes, except ECB, needs an initialization vector (IV) that has to be combined with the original key.

The following subsections presents a set of primitives that could be used for providing security properties in highly constrained devices. These primitives are arranged according to their complexity, and hence according to the resources that they demand to a device. Concretely, they are classified from less to more complex.

2.1.3. Skipjack

Skipjack [Skipjack_1998] is considered one of the simplest and fastest block ciphers, in comparison with other ciphers such as RC5, RC6 or AES. It was designed by the NSA (U.S. National Security Agency), and declassified in 1998. The primitive uses building blocks of 64 bits and a cryptographic key of 80 bits for encrypting data blocks of 16 bits. Therefore, it is especially suitable for constrained nodes with 16-bit microcontrollers, such as the MSP430 microcontroller family.

Skipjack alternates two stepping rules, named A and B, during 32 rounds. Concretely, each rule can be described as a linear feedback shift register with additional non linear keyed G permutation (Feistel cipher of 4 rounds). Also, the key schedule of Skipjack is straightforward, i.e. the key is cyclically repeated enough times to fill the key schedule buffer. Besides these obvious benefits regarding the simplicity of the algorithm, this primitive does not provide the same security than others, due to its small key size.

2.1.4. RC5 and RC6

In 1994, Ron Rivest designed a simple symmetric algorithm named RC5 [RC5_1994]. This primitive has variable parameters both in block size (32, 64, 128 bits), in key size (up to 2040 bits) and in number of rounds (up to 255), although it is recommended to use a block size (b) of 64 bits, a key size of 128 bits and 20 rounds. On the other hand, its internal operations are quite simple: integer additions, bitwise XOR, and variable rotations, over two $^b/_2$ bits registers. An algorithm based on RC5 is RC6 [RC6_1998], which was designed by Ron Rivest, Matt Robshaw, Ray Sidney and Yiqun Lisa Yin in 1998. It is very similar to RC5, except that its recommended block size is 128 bits, and internally includes integer multiplication and uses four $^b/_4$ bits registers.

In spite of using simple building blocks, resulting in a small code size in their implementation, both algorithms are a bit more complex and quite slow, thus not all the highly-constrained devices could support them. Basically, they use 32-bit registers for the recommended block size (i.e. $^b/_2$ where b=64 in RC5, and $^b/_4$ where b=128 in RC6), the number of rounds is a little high, and the key schedule is quite complex in comparison with simpler ones such as in Skipjack. For that reason, the more appropriate microcontrollers are those ones with 32 bits, for example the microcontrollers PXA271 or ARM920T.

2.1.5. AES &Twofish

The Advanced Encryption Standard (AES) [AES2002] was designed in 1998 by Joan Daemen and Vincent Rijmen. It is a derivative of the Rijndael algorithm, which supports a larger range of block sizes ranging between 128 bits and 256 bits. On the other hand, AES works with a fixed block size of 128 bits and a key size of 128, 192 or 256 bits, and depending on the key size the number of rounds is 10, 12 and 14 respectively.

For encrypting, AES needs a 4×4 array of bytes (termed the state) over a finite field. In each round (except the last round) four distinct stages are executed: AddRoundKey, where the subkey is combined (XOR operation) with the state, SubBytes, where each byte is replaced with its entry in a fixed 8-bit lookup table, ShiftRows, where bytes in each row of the state are shifted cyclically toward the left, and MixColumns, where each column is multiplied with a fixed polynomial p(x).

Another block cipher algorithm similar to AES is Twofish [Twofish1999], designed in 1998 by Bruce Schneier, John Kelsev, Doug Whiting, David Wagner, Chris Hall and Niels Ferguson. Twofish uses 128-bit blocks with large key sizes (up to 256 bits), a complex key schedule, and four different key-dependent 8 by 8 bits S-boxes totally bijectives. Besides, it uses other elements belonging to other cipher families, concretely the pseudo-Hadamard transform ($PHT(a,b)=\alpha,\beta$, where $\alpha = a + b \bmod 2^{32}$ and $\beta = a + 2b \bmod 2^{32}$), and the maximum distance separable (MDS) matrix (4×4) over $GF(2^8)$.

Both AES and Twofish are quite fast in their encryption/decryption operations due to their small number of rounds, and also some operations can be calculated in 8-bit registers. On the other hand, they present a considerable complexity, which results in a larger code size. Therefore, not all constrained devices can afford to implement them in software.

2.2. Public Key Cryptography

Public key cryptography (PKC), also known as asymmetric cryptography, is a form of cryptography that uses two keys: a key called secret key, which has to be kept private, and another key named public key, which is publicly known. Any operation done with the private key can only be reversed with the public key, and vice versa. This nice property makes all PKC-based algorithms useful for secure broadcasting and authentication purposes. It is also an invaluable tool for allowing the secure exchange of secret keys between previously unknown partners.

The computational cost of calculating the underlying primitive operations had hindered its application in highly-constrained devices, such as sensor nodes. However, at present there are several PKC primitives that are being considered for a sensor network environment. Obviously, it is important to know their properties to determine if they could be suitable in a constrained architecture. For example, the algorithm RSA [RSA1978] is known for offering good cryptographic operations (encrypting and signing), but limited microprocessors must pay a relatively high computational cost. For all these reasons, it is necessary to analyze what kind of PKC primitives are more suitable for the sensor nodes.

2.2.1. Elliptic Curve Cryptography

In 1985, Koblitz and Miller proposed one of the most investigated PKC primitives in the field of WSN security, named Elliptic Curve Cryptography (ECC) [ECC2000]. This interest is due to some interesting features offered by ECC that are appropriate for WSN context, such as the small key size and fast computation. ECC is based on elliptic curve structures defined as $y^2 = x^3 + ax + b$ over finite fields $\mathbb{F}_p$. Then, the base of security in this type of cryptosystem is determined by the elliptic curve discrete logarithm problem. Concretely, given the parameters a and c, the security is determined by the resolution of the equation $a^b = c$.

The basic operation in ECC is the scalar point multiplication, which can be calculated by the repetitive computation of point additions and doublings. Although point multiplications utilized in the operations of encrypting and signing are less expensive in computational terms than the primitive operations of RSA (i.e. exponentiations), they continue to be resource intensive. For this purpose, there exist several optimizations, such as the use of projective coordinates to avoid the inversion operations or the use of Shamir's trick for minimizing the verification time.

As aforementioned, ECC is able to offer the same security as other cryptosystems, like RSA, providing signatures by using Elliptic Curve DSA (ECDSA) and key establishment by using Elliptic Curve Diffie-Hellman (ECDH). Moreover, ECC works with a smaller key size (160 bits in ECC opposite 1024 bits in RSA), and its basic operation executes in a reasonable time. These properties are very appreciated by the research community in order to try their integration in sensor nodes.

2.2.2. Other cryptographic approaches

Other cryptographic approaches that could be suitable for constrained architectures are Rabin's scheme [Rabin1979], NtruEncrypt [NTRU1998] and *MQ*-schemes [MQ2005]. Overall, their execution time is quite optimal, but their memory requirements in terms of the key size are very heavy.

In 1979, Michael Rabin proposed a high-speed scheme based on the factorization problem of large prime numbers, known as Rabin's scheme. Therefore, its security is determined in the same way as for RSA. This primitive is characterized by the speed of its encryption and signature verification operation, which are built by a simple squaring operation. However, the function of Rabin generates three results: two false and one correct, and the objective is to find the correct one, hence the identification process implicates extra complexity to system. Also, the size of a single signature is quite large (512 bits).

Other asymmetric approach is NtruEncrypt and its corresponding signature scheme NtruSign, which were created by Joseph H. Silverman, Jeffrey Hoffstein, Jill Pipher and Daniel Lieman in 1996. These cryptographic primitives are basically based on a polynomial ring R, and its strength is based on the hardness of solving the Closest Vector Problem (CVP) and the Shortest Vector Problem (SVP). Due to its simple primitive operations for encryption and signing, just mere polynomial multiplications, NtruEncrypt claims to be faster than other asymmetric encryption schemes.

Finally, the most recent asymmetric approach is the multivariate public-key cryptosystems, also known as *MQ*-schemes. Their security is determined by the hardness of solving $w = V^{-1}(z) = (\omega_1,..., \omega_n) \in K^n$, given a quadratic polynomial map $V = (\gamma_1,..., \gamma_m) : K^n \rightarrow K^m$. A *MQ*-scheme is quite fast in its cryptographic operations, such as the verification of a signature. However, there is a significative storage cost for restricted environments due to the size of the keys in RAM. Concretely, the private key needs 879 bytes of storage and the public key needs 8680 bytes. Obviously, this cryptosystem can only be supported by sensor nodes with high memory capabilities.

2.3. Hash Functions

Cryptographic hash functions or hash primitives are utilized in order to compress a set of data of variable length into a set of bits of fixed length. The result is a "digital fingerprint" of the data, identified as a hash value. A cryptographic hash function must satisfy two properties: i) given a hash value *h*, it should be hard to find a message m such that hash(m) = *h*, and ii) it should be hard to find two different messages m_1 and m_2 such that hash(m_1) = hash(m_2).

This kind of hash primitives are usually used to build other cryptographic primitives. For example, the Message Authentication Code (MAC) primitive provides authenticity and integrity in the messages, either by using the CBC mode of operation in a process named CBC-MAC, or by taking advantage of the existence of hash primitives. An example of hash function is SHA-1 algorithm [SHA1_2005], which takes 512 bits and returns 160 bits, and operates with additions, rotations, XOR, AND, OR and NOT. Nevertheless, the complexity required for finding a collision is only 2^{63}, hence it is recommended to use other stronger algorithms such as SHA-256. The algorithm SHA-256 takes 512 bits and returns 256 bits. Other more optimized hash function is RIPEMD-160 [RIPE1996], with an input of 512 bits and an output of 160 bits, which uses rotations, permutations, shifts, and others as internal operations.

Although all the hash functions mentioned in this section are fast, they are only suitable for microprocessors of 32 bits, because of their internal word size. Therefore, they could not be optimally supported by microcontrollers of 8 and 16 bits, whereas more powerful 32-bit microcontrollers like the PXA271 and ARM920T can optimally manage them.

3. Software Implementation Platforms

3.1. Symmetric Key Cryptography and Hash Functions

The task of protecting small, highly-constrained devices is not an easy task, since their microcontrollers are very limited in terms of computational power, memory capabilities, and so on. However, they should be able to execute several instances of Symmetric Key Cryptography (SKC) and Hash primitives without provoking a penalty in communication. That is, the time dedicated in the execution of such software primitives must be under a considerable period of time, because on the contrary it could cause a delay in the sending of packets over the radio. For avoiding this serious problem, the encryption time must be less than the *byte time*, which represents the time required for sending a single byte of data. For example, for a CC1000 transceiver with a bandwidth of 19.2kbps, the byte time is approximately 420µs, and for the 802.15.4-based CC2420 with a bandwidth of 250kps, the byte time is closer to 32µs.

One of the first studies on the encryption overhead of SKC and Hash primitives in constrained microcontrollers was made by Ganesan et. al. [Ganesan2003] in 2003. In their software implementation, they discovered that the time required for encrypting a single plaintext was much less than the time need for executing hash primitives. Nonetheless, the average code size of a single primitive is less than 4000 bytes of ROM. For example, the encryption time of a plaintext with the stream cipher RC4 was 6µs per byte, with RC5 26µs and with IDEA 21µs, whereas for digesting a single byte with the SHA-1 algorithm was necessary to wait 122µs.

Later, in 2004 SKC primitives are introduced by means of a new cryptographic package in Mica and Mica2 nodes over the “de-facto” standard OS for sensor nodes, TinyOS 1.x. This package known as TinySec [Karlof2004] provided the Skipjack primitive, which needed 48µs for encrypting a byte, and the RC5 primitive, which needed 33µs for encrypting a byte. RC5 was considerably better in encryption overhead than Skipjack. However, it did not include any hash primitives, so the integrity process was achieved through use of CBC-MAC, which generates the message authentication code using the SKC primitives.

In 2006, Wei Law et. al. [Wei2006] and Jun Choi and Song [Junchoi2006] carried out a deep analysis on the encryption overhead of other instances of SKC primitives. Their studies demonstrated that an optimized Skipjack improved in both encryption overhead (25µs) per byte and in memory overhead (2600 bytes of ROM) than the rest of algorithms (an average of 8000 bytes). Nonetheless, RC4 achieved in terms of memory overhead 428 bytes, thus it was much better than an optimized Skipjack for extremely constrained devices.

3.2. Public key Cryptography

Until 2004, almost all security studies were focused on symmetric key cryptography (SKC), since the possibility of using public key cryptography (PKC) in highly-constraint context (for example, Wireless Sensor Network (WSN)) was considered, even labelled, as "not possible". However, Gura et. al.'s studies in [Gura2004] opened a door of possibilities for the applicability of PKC in sensor networks. They ensured that Elliptic Curve Cryptography (ECC) was an appropriate technique for implementing PKC in high-constrained context, since it offers a good functionality both in computation and memory storage, minimizing energetic costs and functional complexity. The cause of this reduction is due to its small key sizes and its faster computation.

Moreover, Gura et. al.'s studies encouraged to the research community to advance more on this topic. In fact, Malan et. al. [Malan2004] presented in the same year the first library with ECC primitives over the field $F_2{}^p$, known as EccM 2.0. It worked with a key size of 163 bit, achieving an average execution time of 34 seconds in its operations (point multiplication and secret key generation). However, 34 seconds was not fast enough for allowing the creation of secure services based on PKC. For that reason, several studies started to optimize the ECC capabilities. One of the firsts ECC optimizations was presented by Gura et. al. in [Gura2004]. They ensured that performance of ECC could be improved if it used projective coordinates instead of affine coordinates to reduce the number of expensive operations, such as the inversion. Other of their suggestions was to work over a field F_p instead of $F_2{}^p$. Such optimizations were taken into account by Batina et. al. [Batina2006] for their work on their hardware implementation of ECC (cf. Section 5.1.1).

Actually, there are several implementations using a few of the ECC optimizations previously described. For instance, Liu and Ning implemented TinyECC [Liu2007], and Wang and Li implemented WMECC [Wang2006]. Concretely, TinyECC has a set of implementations in various elliptic curve domains (secp128r1, secp128r2, secp160k1, secp160r1, secp160r2, secp192k1, and secp192r1) according to the Standards for Efficient Cryptography Group [SECG2007], while WMECC only has the implementation on the secp160r1 elliptic curve domain. All of them work under an event-oriented and component-based Operating System named TinyOS. They were codified with a component-oriented language created specifically for sensor networks, named nesC (Network Embedded System C).

Although, TinyECC and WMECC were very different both in design and in code, they were relatively similar because they have in common some optimization techniques. Firstly, for getting optimization in the modular reduction in multiplications and square, they made use of pseudo-Mersenne primes (being p a field prime, whose value is $2^n - c$). Nonetheless, TinyECC is a little better in performance than WMECC on this aspect. It can compute modular multiplication of large integers by using Hybrid Multiplication. Secondly, both implementations used projective coordinates, but TinyECC applied Jacobian representation (X,Y,Z) while WMECC utilized a combination of representation between modified Jacobian coordinates (X,Y,Z,aZ^4) and Affine coordinates (X,Y).

Lastly, both implementations utilized two methods, Shamir's trick and sliding window method, for improving the performance of scalar point multiplications. The purpose of the Shamir's trick is to execute in parallel several point multiplications, and this way it is possible to minimize the signature verification time. The sliding window method, grouping the scalars in *s*-bit clusters, allows the reduction of the running time by pre-computing point multiplications. It is important to notice that the pre-computation phase in TinyECC is much more expensive than WMECC, since it must initialize the ECDSA system. On the contrary, WMECC uses a small window for both methods (*s=4* for sliding window method and *w=1* for Shamir's trick).

In order to analyze discrepancies between both implementations, it is necessary to compute several rounds of them for determining which is their overhead and complexity cost. Table [X] shows the results obtained by the execution of TinyECC and WMECC in the elliptic curve domain secp160r1. This table summarizes the results obtained after executing 20 rounds of the ECC primitives over the Micaz and TMote Sky nodes. The first rows of the table corresponding to the ROM and RAM size reserved for the primitives and the test program, and the rest of rows corresponding the time spent in the execution of the primitive. The difference between both nodes can be partially explained by their architecture: the microcontroller of Micaz does not spend time in computing unsigned multiplication, but the microcontroller of TMote Sky (MSP430) dedicated several cycles to them.

Comparing both implementations, the WMECC package yields 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, the major penalty of the WMECC code is the SHA-1 function, and the TinyECC implementation is precisely characterized by an optimized SHA-1 code. Thanks to the component capabilities of the TinyOS operating system, it is possible to use the SHA-1 component of TinyECC in the WMECC code, resulting in a new version of the WMECC package 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.

Summing up, the results obtained for TinyECC and WMECC achieved the expected optimizations, reducing the running time for signing or verifying a signature to not more than 2 seconds. This indicates that there was an important improvement with respect to the results obtained by Malan et. al.'s work. As a result, it has been shown that it is possible to use software implementations of PKC in constrained sensor nodes. On the other hand, it would be better to reduce even more the execution time of these primitives for allowing the creation of efficient secure services based on asymmetric cryptography.

4. VLSI Integrations for Cryptographic Primitives

The applications increasing demand for computation power, and the power reduction requirements for portable devices, force researchers to consider that general-purpose processors are no longer an efficient solution for mobile systems. So, new hardware approaches are needed in order to implement some computational heavy and power consuming functions in order to meet the current network speed requirements. Such approaches are:

Recent Application-Specific Integrated Circuits (ASIC) technology was the solution that created better opportunities for implementing real-time and more sophisticated systems. ASIC devices guarantee better performance, with enough small dedicated size. The reliability reaches high limits and the performance reaches high values. The implementations in these modules are characterized of tighter design stability, than any other type of hardware devices. ASICs include several custom and semi-custom hardware designs such as: Programmable Logic Devices (PLD), Gate Arrays (GA) and Standard Cells (SC). In our case ASICs can be described as follows: Custom-designed hardware, specially tailored to one particular encryption process. They require a significant initial investment for design and testing. If such a device is not produced in mass quantities, it is not economical for the market. ASICs seem to be more suitable for dedicated applications and not for an extended purposed encryption system.

Besides the original software implementation platforms and the ASICs devices integrations there is a middle ground. This area is covered by the Field Programmable Gate Arrays (FPGAs) and Complex Programmable Logic Devices (CPLDs). These components provide reconfigurable logic and they are commercially available at low prices. They facilitate hardware/software codesign. Of course, these devices vary in capacity and performance. Programmable logic has several advantages over custom-hardware. It is less time-consuming, for the development and the design phase, than the custom-hardware approach. These devices are more flexible than ASICs. They can be reused for cryptanalysis of many different encryption algorithms with little extra effort.

Another solution to the implementation platform problem is smart cards. This issue has to do more with fit than with performance. In smart cards the RAM requirements are more important, than the clock's frequency. Most commodity smart cards CPU today include 128 - to 256 - bytes of on-board RAM. Each CPU family contains members with more RAM capacity and a correspondingly higher cost. Although some CPUs include enough RAM to hold the keys of the algorithms, it is often not realistic to assume that such a large fraction of RAM can be dedicated solely to the encryption function. In a smart-card operating system, encryption is a minor small part of the system and will not be able to use more than half of the available RAM. Obviously, if an encryption/decryption system does not fit on a certain CPU, with particular configuration of its components, the performance of the system is unrealistic. Even if an algorithm fits onto smart card, the encryption function will not be able to use all of the RAM capacity. For example, an algorithm that needs at about 100-bytes RAM seems to fit in a 128-bytes smart card. Of course this is a theoretical result because there are RAM requirements also for the control procedure that handles the total security process and these requirements increase the need of the memory-limited capacity. It is cleared that the devices of this category are not proper for large encryption systems with special specifications.

In general, hardware implementations have been proved better approaches compared with the software developments, in the terms of throughput, and operating frequency. Of course the covered area resources is a factor that have to be under consideration.

For all the hardware devices there are some common factors that make the implementation of the ciphers in powerful hardware engines a very hard process. The most critical of them is the large number of registers for key storage, which are used by most of the algorithms. As it has been already mentioned above, the security of ciphers is a function of two factors: the strength of the algorithm and the length of the key. While the strength of a cipher is a fixed factor since algorithm's definition, the key length is a parameter that can vary. Ciphers' introducers and cryptographers use large keys for more secure operations. This means larger number of buffers and storage units and increased memory requirements, for hardware integration. This event has finally cost in the chip's covered area and sometimes in the I/O devices of the system. In order to face this problem RAM blocks are mainly used in hardware implementations. However, in many cases the availability of RAM is restricted. The internal memory capacity of many hardware devices is limited. The use of external RAM reduces the total system performance and increases the system's covered area. All these factors are critical items, which must especially be taken into account by the designers. The application itself defines each time the impact grade of these factors.

5. Integration on Silicon for WSN

5.1. Existing Efforts

5.1.1. Research Solutions

The sensor network paradigm is still in its infancy, and the efforts in both research and industrial environments have been focusing on the design and development of sensor node hardware platforms. On the other hand, there have been few advances in the area of security-specific hardware for sensor networks, and they have not surpassed the stage of prototypes. A possible reason could be the lack of real-world applications beyond test beds. Although it is theoretically clear that in this kind of network environments there is a need for security primitives, mainly due to the public nature of the communication channels, there are no existing applications that could force the creation of specific hardware integrations at an industrial level.

Most of the existing prototypes, which have been designed on research environments, are specialized on providing Public Key Cryptography primitives. These primitives are extremely resource-demanding for highly constrained nodes due to the complexity of their internal operations, compared to the other primitives like Symmetric Key Cryptography. As seen in section 3, the memory footprint of a software implementation of a PKC primitive is usually quite high, consuming more than $^1/_3$ of the memory of a node on certain platforms, and the execution time of a simple operation such as signature verification can last at least 2 seconds. A hardware implementation could greatly improve the usability of these primitives, enabling the creation of advanced authentication services.

Due to its benefits in key size and complexity, Elliptic Curve Cryptography (ECC) has been the primitive of choice in most asymmetric cryptography hardware prototypes as of 2007. The main purpose of these implementations is to reduce the time on executing the ECC core operation, the point multiplication, from the seconds in software implementations to just hundred of milliseconds. A majority of the prototypes are specialized on achieving an acceptable performance using as little gates as possible while functioning with an operational frequency smaller than 8 Mhz, which is the usual operational frequency for a normal sensor node.

In fact, the prototypes that have been specifically created for sensor nodes seek the provision of ECC operations to both normal and "weak" nodes, using an operational frequency as small as 500 Khz. An example is the design by Gaubatz et. al. [Gaubatz2005], where 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 133μA in signing and encrypting messages. ECDSA signatures and ECMQV decryptions are performed at around 410ms, and ECDSA verifications and ECMQV encryptions are performed at around 820ms. As for the specific optimizations in this design, they efficiently implement modular reduction, achieve a fast inversion algorithm, and implement all arithmetic primitives in a bitserial fashion.

A later prototype, created by Batina et. al. [Batina2006], improves the results obtained in the previous design using the same operational frequency, 500 Khz. 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 (12000 gates including RAM) in 0.13μm CMOS technology, one point multiplication over $\mathbb{F}_{2^{131}}$ is performed in only 115ms, consuming less than 10μA.

On the other hand, not all ECC prototypes consider the operational frequency as a factor that should limit the design of the hardware. The existence of "heavy-duty" microcontrollers used on certain sensor node platforms, like the PIC18F6720, PXA271, and the ARM920T, justify the creation of specialized prototypes that operated at a higher frequency. Wolkerstorfer et. al. [Wolker05], as well as Kumar and Paar [Kumar2006], designed integrated chips that are able to provide an excellent performance for signing a message using ECDSA. In the case of Wolkerstorfer et. al., the chip, which has an area of 23000 gates implemented in 0.35μm CMOS technology, is able to execute a single point multiplication operation over $\mathbb{F}_{2^{191}}$ in only 6.67ms. This design has an operational frequency of 68.5 Mhz, thus it can only be used by "heavy duty" sensor nodes. In contrast, the chip created by Kumar and Paar is slower, providing a point multiplication over $\mathbb{F}_{2^{131}}$ in 18ms, but the number of gates is smaller, almost 12000. Its operational frequency is also smaller, around 13 Mhz, so it can be afforded by "heavy duty" and normal microcontrollers alike.

There are also other hardware prototypes that provide other asymmetric cryptography primitives, such as NTRU, Rabin, and Multivariate (cf. Section 2). The primary disadvantage of these primitives, compared with ECC, is their key and signature size. The Multivariate cryptosystems uses a private key of 879 bytes, and a public key of 8680 bytes. Also, NTRU provides a signature of 1169 bits, while Rabin generates a signature size of 512 bits. As a result, there is an energy overhead associated to the transmission of messages over the communication channel and a cost associated to the storage of keys and signatures. All of these are critical factors in the design of protocols for sensor networks. Nevertheless, these prototypes have certain advantages that can make them useful on certain scenarios.

The NTRUEncrypt prototype, developed by Gaubatz et. al. [Gaubatz2005], employs a very small number of gates, around 3000, and consumes only 6.6μA 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. As a result, most operations done on a NTRU chip are faster than any of the ECC prototypes. Also, their chip area is the smallest of all the PKC primitives. The major benefit of the Rabin prototype, which was developed by the same authors [Gaubatz2005], is the encryption and verification operations: at the same operational frequency of the NTRU prototype, 500 Khz, a message can be encrypted or verified in just 2.88ms. Still, the downside comes from the decryption and signature operations, which are calculated in 1.089s, and from the average power consumption, which is near 49µA.

Finally, the Multivariate prototype, made by Yang et. al. [Yang2006], only provided signature generation, since it was designed for being included inside RFID tags. Nevertheless, since the target device is even more constrained than a sensor node, the benefits are higher in comparison with the other prototypes: using 17000 gates in 0.25µm CMOS technology, with an energy consumption of 25µA in an operational frequency of just 100kHz, it is possible to sign a message in 44ms. This speed is achieved by using the most optimal primitive, the circulant (LPSQR) form of enTTS(20,28), in conjunction with some optimizations like the substitution of the Lanczos' method used in the internal operations with the symmetric Gaussian elimination.

5.1.2. Industrial Efforts

Aside from the previously mentioned prototypes developed on research environments, there are, at present, very few solutions that are able to provide hardware support for cryptographic primitives at a mass scale. The most effective solution in terms of using the primitives would be to define a standard for including this kind of hardware support in the microcontrollers. However, there are no existent solutions in this area, aside from the definition of processor-specific instructions sets like MMX that could be used to improve the execution of the security primitives. On the other hand, some network standards used for wireless communication in sensor networks demand the existence of support for such primitives.

As a result, on current technology, the radio transceiver chip located on sensor nodes can be able to provide hardware support for cryptographic primitives. The reason is simple: usually, all communications have to be protected with symmetric key cryptography primitives in order to assure the confidentiality, integrity and authenticity of the packets that traverse the network. Since all network packets must traverse the radio transceiver, this is the ideal place where the hardware support could be located. Not all radio transceivers incorporate this type of support, though: only a certain group of standards requires the support of the primitives on silicon. One of those standards, widely used on sensor networks, is the IEEE 802.15.4 standard [IEEE802.15.4].

All transceiver chips that implement the 802.15.4 standard can offer a suite of AES-based symmetric key cryptography operations for assuring the confidentiality and integrity of the network packets. The available suites are AES-CTR, AES-CBC-MAC, and AES-CCM. The AES-CTR suite only provides confidentiality by using AES in Counter Mode, thus behaving as a stream cipher. AES-CBC-MAC ensures data integrity through Message Authentication Codes using AES in Cipher Block Chaining Mode (CBC-MAC), where the size of the MAC can be 32, 64 or 128 bits long. Finally, AES-CCM provides both confidentiality and integrity, by applying integrity protection using AES-CBC-MAC and then encrypting the data using AES-CTR mode.

In order to use these security suites, the application must create an ACL (access control list) entry consisting of 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 standard-compliant radio transceiver receives a packet with security enabled, it looks up 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 transceiver 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 [Sastry2004]. 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 entries when entering low power mode. As a result, chip manufacturers and application designers must take into account the inherent problems of the standard.

As an example, the most common 802.15.4 chip integrated in sensor nodes as of 2007, 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 [Sun2006].

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.

These extended instruction sets were not designed with cryptographic primitives in mind, since they are focused on providing support for multimedia-based tasks such as video compression, 2D and 3D graphics, and image processing. Nonetheless, their parallelism capabilities can help to efficiently implement block ciphers such as AES or Twofish [Aoki2000], 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 Pseudo-Hadamard Transformation in Twofish somewhat complicates the parallelization process, although it is still possible to optimize certain parts of the algorithm.

Not only block ciphers can be optimized: hash functions can be also improved by the use of MMX instructions. For example, Nakajima and Matsui [Nakajima2002] explored the optimization of MD5, the RIPEMD family, the SHA family and Whirlpool, by processing message blocks in parallel using MMX registers for 32-bit oriented hash functions. This type of parallelization can be possible in embedded systems, but it has not been researched yet. There were also some optimizations for 64-bit oriented hash functions like SHA-512 and Whirlpool, but these optimizations cannot be applied to the “Wireless MMX” instruction set, because it cannot parallelize operations over 64-bit fields.

6. Alternative Solutions and Optimizations

The problem of hardware implementation is a function of two different factors: cryptographic algorithms architectures and the efficient integration of them [Sklavos2007]. All forums and organizations in the wireless communication world have specified security layers/systems and have published the selected ciphers that these systems are based on. In order high-level security to be ensured, three schemes of encryption must be applied in a communication handshake: Bulk Encryption, Message Authentication and Data Integrity. The wireless protocols have defined alternative ciphers in each type of the above schemes. Large encryption systems have been mainly implemented only in software. On the other hand, hardware devices have been used for the implementation of encryption algorithms, one per device, and for security systems with less complexity.

The previous years, the hardware integration approach to the issue of security implementation was the ASICs solution. Implementations on these modules achieve high-speed performance and have been proved confidant solutions. Although in the case of wireless protocols, this implementation aspect is proved unfeasible. The hardware integration of a set of ciphers, that a protocol defines, results in a very large circuit. Encryption algorithms implementations, that have been published until now in ASICs, cover an area of 40-60 mm2 each. For example, the WAP cipher set integration (eight algorithms in total), in one or more ASICs needs an area about 400-480 mm2, plus the space needed for the total control unit and routing allocated area. Such an ASIC device is very difficult to be designed and manufactured. Of course the cost of the chip is increased dramatically in this case.

Nowadays a flexible encryption system, which would support the operation of a set of ciphers integrated in the same module, can be implemented with hardware and software cooperation. This type of cooperation could be achieved efficiently by the principles of reconfigurable computing. A proposed solution is the design of a reconfigurable cryptographic system, which will support at least bulk and message authentication encryption. Reconfigurable computers are those machines that use the reconfigurable aspects of Reconfigurable Processing Units (RPUs) and FPGAs to implement a system/algorithm. The algorithms are partitioned into a sequence of hardware implementable objects (hardware objects). This type of objects represents the serial behavior of the algorithm and can be executed sequentially. The design technique based on hardware objects offers to the developer/designer a logic-on-demand-capability that is based on the reconfigurable computing. The appropriate software, in order to suit the application at hand, modifies the architecture of these computing platforms. This means that within the application program a software routine has been written to download a digital circuit (chip design) directly into the RPU. The main idea of these designs is the alternation among static and dynamic performance of the system.

Static circuitry is the part of the operation performance that remains in action between the different configurations of the hardware device. The part of static circuitry has to be in the maximum level of the design and special care must be taken for its optimization. General-purpose blocks, such as adders, belong to this part of performance. Another example of the static parts is the storage units. These are the parts of each system that never are never changed during different operations. Always they maintain the characteristics that the initialization process has set. On the other hand, there is the dynamic circuitry. With this term, we mean the parts of the system that change during configuration. These blocks must be minimized in order to increase the system performance. If there are not basic common parts between the selected algorithms, the dynamic circuitry reaches high values and this is not flexible for the system performance. Dynamic circuitry increases the needs for the system's allocated resources, and on the other hand decreases its attribution. It has to be cleared that the selection ciphers with similar philosophy of functionality, finally would be proved as a critical factor of the hardware implementation.

Although, in most of the case encryption algorithms are based in different design philosophy and their functionality is completely different. In order to achieve this, the designer of a powerful security system has to choose one flexible algorithm for bulk encryption with the ability to operate as a hash function to support data integrity purposes, for example HMAC. The addition of some extra parameters in the algorithm's architecture is necessary for the efficient operation of the two encryption modes. In this way, the needs of the system resources are reduced. At the same time, we have to avoid ciphers with heavy arithmetic functions such as multiplication and modulo processes. Such arithmetic functions are proven quite difficult to be implemented in hardware devices and mainly they have no common parts with other ones.

The implementation of a security system with some common basic parts, which can be used for the implementation of ciphers' common functions, seems to be the more sophisticated alternative solution for a large encryption engine. With the term basic parts we mean "heavy" algebraic or logical components of the algorithms' architectures. In most of the cases it is difficult to implement these parts in a hardware device, with high-speed performance and minimized covered area. An example is the multiplication modulo that IDEA cipher needs. Reconfigurable computing method is proved efficient enough to solve the implementation problem of encryption engines and is suitable for the different types of architectures that ciphers' have.

The latest years, implementations on the smart card devices have been very attractive for the hardware designers. Compared with the other hardware devices like ASICs and FPGAs, smart cards have limited computing power and minimized storage capacity. Therefore, security applications which allocate a huge amount of storage or which require an extensive computation power might cause conflicts.

The persistent storage of a smart card is limited to a few kilobytes today, which prevents it from storing larger items on the card. This can be circumvented if the smart card delegates the storage of the item in an external environment. The smart card receives and processes the transmitted data. It encrypts them and saves them to the sender/receiver's device external storage units (RAM, registers of general use). Later, when these data are needed again, the smart card can request it from these storage units. By using this described method the smart card internal storage requirements can be reduced significantly. However, we have to take care that we do not create another bottleneck: the communication speed of the smart card is not very high and so we should have to handle the transmission of the same data back and forth, with special care.

Another limitation of smart cards is the small processing power. The appropriate data modifications, due to encryption/decryption, may possibly exceed the computing power of a smart card. In this case it will take unacceptably long to finish the appropriate data transformation. Thus, it is important to minimize the amount of computation power, which the smart card has to pay for the requested tasks. For such applications, it is better the design to be kept as simple as possible. The requested task can be divided in smaller parts with no hard processing specifications. The requested round keys for encryption/decryption can be generated in the initialization procedure and not at the same time with the encryption round transformation (on the fly key generation). In this way, we avoid to spend extra processing power for the key expansion unit during encryption/decryption. The same methodology can be followed for the appropriate specified constants generation.

7. Future Directions

The needs for personal network communications systems are growing rapidly. Coupled to this increase is the telecommunication-related crime. In wireless networks, an invader with suitable receiver can intercept the transfer data. Security is a primary requirement of all wireless cryptographic protocols. Cryptographic algorithms are meant to provide secure communications applications. However, if the system is not designed properly, it may fail. Although there are many well know ciphers, with different specifications and characteristics, the security of some of them is under consideration. Many works, from different research groups, have been published in technical literature in which cryptanalysis methods have been applied in order to find out any existing black holes in the security strength of the encryption algorithms. From many points of view, such attempts offer valuable knowledge in the growth and the improvement of cryptography. Encryption algorithms have to perform efficiently in a variety of current and future applications, doing different encryption tasks. The algorithms should be used to encrypt streaming audio and video data in real time. They would have to work correctly in 32 and 64-bit CPUs. Many of today's applications run with smart cards based on 8-bit CPUs, such as burglar alarms, pay-TV and general other applications. All hardware implementations have to be efficient, with less allocated area resources. This means simplicity in algorithm's architectures with enough "clever" data transformation components. A wireless protocol implementation demands low power devices and fast computation components, which imply that the number and complexity of the encryption operations should be kept as simple as possible. A basic transformation in the operation of the encryption algorithms is needed, including modifications in the data blocks and key sizes.

The ciphers of the future have to be key agile. Many applications need a small amount of text to be encrypted with keys that are frequently changed. Many well know applications, like IPsec, use this way of algorithm's operation. Although the most widely used mode of operation is encryption with the same key for all the amount of transport data, the previous mode is also very useful for future applications. Ciphers that require subkeys precomputation have a lower key agility due to the computation time, and they also require extra RAM to hold the subkeys. This RAM requirement does not exist in the implementations of algorithms, which compute their keys during the encryption/decryption operation. Cellular phones technology demands hard specifications of the cryptography science. Ciphers have to be compatible with wireless devices restricted standards in hardware resources. New mobile phones will have proper encryption part built in them. In these devices, there is not enough room for a large integrated security layer. A solution in order to decrease the required hardware resources is to use the ciphers for both bulk encryption and data integrity, with a simple change of their operating mode. All the above, make the effort of designing new security algorithms for wireless applications a real hard process. Both AES and SHA-2 standards are very good steps for the design of communications security schemes, for the next decades.

The technology growth gives many promises for the security future. If the strength of the applied cryptography that is used in wireless industry were increased enough, the protocol security would be efficient to withstand the attempts of the attackers. Today many ciphers can support the defense of the communications links against external invaders. On the other hand, the implementation of these is a hard process and sometimes cannot meet the wireless network requirements. This is due to the fact that that today's used ciphers have been designed some years ago and for general cryptography reasons. They are not specialized for the wireless communications. Security improvement needs strong, flexible encryption algorithms with efficient performance. New encryption algorithms must be designed for applications of any kind. On the other hand, every algorithm should be demonstrated in software before committing to hardware.

8. References

[AES2002] J. Daemen, V. Rijmen. "The Design of Rijndael". Springer, ISBN 3-540-42580-2, 2002.

[Aoki2000] K.Aoki and H. Lipmaa. "Fast implementations of the AES candidates". Proceedings of 3rd AES conference, New York (USA), April 2000.

[Batina2006] 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.

[ECC2000] I. Blake, G. Seroussi, N. P. Smart. "Elliptic Curves in Cryptography". Cambridge University Press, ISBN 0-521-65374-6, 2000.

[Fluh2007] S. Fluhrer, I. Mantin, A. Shamir. "Weaknesses in the Key Scheduling Algorithm of RC4". In proceedings of the 8th Annual Workshop on Selected Areas in Cryptography (SAC 2001), Toronto (Canada), August 2001.

[Ganesan2003] 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.

[Gaubatz2005] 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.

[Gura2004] 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.

[IEEE802.15.4] IEEE 802.15 WPAN^{TM} Task Group 4 (TG4). http://www.ieee802.org/15/pub/TG4.html

[Junchoi2006] 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.

[Karlof2004] 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.

[Kumar2006] S. Kumar, C. Paar. "Are standards compliant elliptic curve cryptosystems feasible on RFID?". In Proceedings of Workshop on RFID Security, Graz (Austria), July 2006.

[Liu2007] 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.

[Malan2004] 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.

[MQ2005] C. Wolf, B. Preneel. "Taxonomy of Public Key Schemes Based on the Problem of Multivariate Quadratic Equations". Cryptology ePrint Archive, Report 2005/077.

[Nakajima2002] Junko Nakajima, Mitsuru Matsui. "Performance Analysis and Parallel Implementation of Dedicated Hash Functions". Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques: Advances in Cryptology (Eurocrypt 2002), Amsterdam (Holland), April-May 2002.

[NTRU1998] 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.

[Rabin1979] M. O. Rabin. "Digitalized Signatures and Public Key Functions as Intractable as Factorization". Technical Report MIT/LCS/TR-212, Massachusetts Institute of Technology (1979).

[RC4_1996] B. Schneier. "Applied Cryptography, 2nd edition". Wiley, ISBN 0-471-12845-7, 1996.

[RC5_1994] R. L. Rivest. "The RC5 Encryption Algorithm". Proceedings of the 2nd International Workshop on Fast Software Encryption (FSE 1994), Leuven (Belgium), December 1994.

[RC6_1998] 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/

[RIPE1996] 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.

[RSA1978] 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. 120–126, 1978.

[Sastry2004] 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.

[SECG2007] SECG - Standards for Efficient Cryptography Group. http://www.secg.org/

[SHA1_2005] 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.

[Skipjack1998] NIST-CSRC. “SKIPJACK and KEA Algorithm Specifications, version 2”. 29 May 1998, http://csrc.nist.gov/CryptoToolkit/

[Sklavos2007] N. Sklavos, X. Zhang, Handbook of Wireless Security: From Specifications to Implementations, CRC-Press, A Taylor and Francis Group, ISBN: 084938771X, 2007.

[Sun2006] 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.

[Twofish1999] 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.

[Wang2006] 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.

[Wei2006] 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.

[Wolker2005] J. Wolkerstorfer. “Scaling ECC Hardware to a Minimum”. In ECRYPT workshop - Cryptographic Advances in Secure Hardware - CRASH 2005. Leuven (Belgium), September 2005. Invited Talk.

[Yang2006] 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.