Optimization of Public Key Cryptography (RSA and ECC) for 16-bits Devices based on 6LoWPAN

Jesús Ayuso and Leandro Marin
Department of Applied Mathematics
Computer Science Faculty
University of Murcia
Murcia (Spain)
Email: {jap89826, leandro}@um.es

Antonio J. Jara and Antonio F. Gómez Skarmeta
Department of Information and Communications Engineering
Computer Science Faculty
University of Murcia
Murcia (Spain)
Email: {jara, skarmeta}@um.es

***Abstract*—Internet of things (IoT) is one of the last advances in ICT, providing a global connectivity and management of sensors, devices, users and information. Specifically, we are working on Future Internet devices based on 6LoWPAN. 6LoWPAN presents an extension of WSNs to the Internet, which has defined a new set of security challenges. For example, new cryptographic mechanism more scalable are required, in order to provide security properties such as authentication, privacy and integrity to communications end-to-end trough the Internet. Right now, WSNs cryptographic is based on Symmetric Key Cryptography (SKC), but it is not suitable for IoT, since symmetric key establishment is not scalable. Fortunately, research on security for sensor networks is showing promising results such as efficient implementations of Public Key Cryptography (PKC) algorithms and lightweight self-healing mechanisms.**

Our contribution is an analysis of security requirements from IoT applications, and mathematical optimization for RSA and ECC algorithms in 6LoWPAN nodes. Specifically, multiplication operation is the most expensive operation in RSA and ECC algorithms, i.e. it consumes the majority of the time, since it is repeated thousands of times. Therefore, multiplication operation is the part that will be discussed in more detail. This work proposes an implementation of multiplication operation for RSA and ECC based on bit shifting instead of the microprocessor's multiplication operation. It seems a contradiction to study the process of bit shifting, when exists a specific operation, but we have found that in 6LoWPAN devices, such as Tmote Sky and rest of motes based on MSP430 microprocessor, multiplication operation is not supported, it is actually emulated and consequently requires a big amount of clock cycles (150 cycles in MSP430), while the bit shifting is supported and it just needs between 1 an 4 cycles.

The evaluation has presented that bit shifting is better than microprocessor's multiplication operation with a relation of $32\alpha k^2 < 3(\mu+\alpha)k^2 + 2(2\mu+\alpha)k$, where α is the number of cycles for bit shifting, and μ is the number of cycles for the microprocessor's multiplication. Therefore bit shifting is more suitable when its cost is 15 times or less than multiplication i,e, $\mu < 15\alpha$. MSP430 has a μ/α between 38 and 150.

I. Introduction

Providing effective and appropriate security primitives is one of the most important objectives of information and communication technologies (ICT). Internet of things (IoT) is one of the last advances in ICT, providing a global connectivity and management of sensors, devices, users and information. Particularly, 6LoWPAN technology offers the capacity to extend WSNs based on IEEE 802.15.4 to Internet. 6LoWPAN is an open standard for low power WPANs under IPv6 [10]. This standard is considered a suitable element to introduce the concept "Internet of Things" in the real-world [21].

IoT technologies offer a wide range of applications in healthcare, which improves the quality of services, reduce mistakes, and even detect health anomalies from vital signs. This paper presents briefly how IoT technology is applied in clinical environments to present the requirements from an IoT environment for security, privacy and authenticated access.

The problem is that 6LoWPAN nodes are highly constrained in terms of computational capabilities, memory, communication bandwidth, and battery power. As a result, it is challenging to implement and use the cryptographic algorithms and protocols required for the creation of security services.

Our research is focused on cryptographic primitives, which are required by most of the security protocols and mechanisms in order to integrate the security properties into their operations. These cryptographic primitives are Symmetric Key Cryptography (SKC), Public Key Cryptography (PKC), and Hash functions. SKC and Hash functions are directly supported by IEEE 802.15.4. The problem is that IoT requires a higher scalability, since any node from Internet should be able to connect with a node from a 6LoWPAN. For that reason, PKC needs to be supported in 6LoWPAN. Our research in this paper has been evaluate and optimize RSA and ECC algorithms for 6LoWPAN nodes, such as Tmote Sky.

Therefore, the purpose of this paper are: Firstly, to analyse the impact of the integration of WSN and the Internet, this is considered a hypothetical clinical scenario based on 6LoWPAN, over WSNs and over that clinical scenario is analysed the security threats and requirements, in Section II. Secondly, it is carried out an overview of the different security mechanism for sensor networks, in Section III. Finally, we propose a set of mathematical optimizations to do feasible the application of Public Key Cryptographic in 6LoWPAN nodes, in Section V, which are being evaluated in Section VI.

II. Security Threats and Requirements in Future Internet networks based on 6LoWPAN

A. Security Threats

This section presents a brief overview of the different threats, which WSN are vulnerable. A deep analysis of them can be found in [20]. The security threats types are:

- **Common Attacks:** Wireless communication channel is public, therefore it is subject to various types of attacks, either passive (eavesdropping) or active (data injection).
- **Denial of Service Attacks (DoS):** The objective of this attack is to avoid the provisioning of services. The most basic DoS attacks can target the nodes themselves (power exhaustion attack), e.g. sending request continuously until the node consume its battery, or the communication channel (jamming attack), e.g. introducing noise signal in the frequency which node is transmitting.
- **Node Compromise:** An device is considered being compromised when an attacker is able to either read or modify its internal memory, and access to the node information such as secret keys or confidential data.
- **Side-channel Attacks:** An adversary can monitor certain physical properties of the nodes, such as electromagnetic emanation from the chip's surface in a passive way, in order to extract the information of the device.
- **Impersonation Attacks:** An attacker node can create multiple fake identities (sybil attack), and also can create duplicates with the same identity (replication attack).
- **Protocol-specific Attacks:** Some essential protocols used in WSN, such as routing, aggregation, and time synchronization, are targeted by some attacks in order to influence the internal nodes functionality and the network services.

B. Security Requirements

Such as presented in the Section II-A, WSN are highly vulnerable to attacks. The effects of those attacks are not desirables. For example, following the hypothetical scenario of a clinical environment, a DoS attack could stop a critical patient continuous vital sign monitoring , consequently in case of anomalies, they are not notified. For example, impersonation attack could reply information from a patient, e.g. informing that he is right when he is not. Finally, any access to the patient's information means access to the patient's privacy. Therefore, it is clear the need of security mechanisms to prevent the attacks and to minimize the adverse effects of such attacks. The desirable security properties, following [20], in a WSN are:

- **Confidentiality:** A message should be understood just by the recipients. Confidentiality is one the most important security properties in clinical environments, since patient's information is private and sensitive. It may be not mandatory in certain scenarios where the data is public (e.g. environment data such as humidity).
- **Integrity:** A message should not be altered or modified. Integrity is a mandatory property because the trust in the information is related with the trust in the integrity of this. Since wireless medium is open, it is easy to get a packet, manipulate it and forward it with a modified content. For example, in clinical environments an attack could change the glucose value of a patient.
- **Authentication:** Data authentication allows a receiver to verify that the data is really sent by the indicated sender. This security property is quite important in sensor networks. In clinical environments it is really important, since the patient's node needs to authenticate to the sender of the private information request.
- **Authorization:** This property ensures that only authorized entities are able to perform certain operations in the network. For example, in clinical environments it is necessary to ensure that just some doctors have a proper authorization, in order to perform certain tasks and access to specific private information.
- **Availability:** The users of a sensor network must be capable of accessing its services whenever they need them. Therefore, the different elements of the network must be robust enough to provide services even in the presence of malicious entities or adverse situations. In clinical environments, this security property is necessary to ensure that patient's data and alarms are delivered.
- **Freshness:** Data freshness means that the data produced by the sensor is recent. In clinical environments, the information must be received as soon as possible (e.g. alerts of heart attack). Freshness is also linked to duplicated information, i.e. when an adversary success on replaying an old message inside the network.
- **Forward and Backward Secrecy:** Since new sensor nodes are deployed whenever other sensor nodes fails to replace it, appear two new properties that need to be considered, on one hand, forward secrecy, where a sensor should not be able to read any future messages, and on the other hand, backward secrecy, where a joining sensor should not be able to read any previous message.
- **Self-Organization:** Sensor nodes should be independent and flexible enough to autonomously react against problematic situations, organizing and solve the problem themselves. These problematic situations can be caused either by attackers trying to influence over the functionality of the system elements and by extraordinary environmental circumstances in the network.
- **Auditing:** Sensor network should be able to log significant events that occur inside the network. This property is necessary due to the autonomous nature of the nodes, since users are not controlling their functionality, they are not able to know about the existence of a certain event unless the nodes log it.
- **Non-repudiation:** A node cannot deny sending a message it has previously sent. For achieving it, it is necessary to produce certain "evidence" in case a dispute arises. Using the evidence, it is possible to prove that a device of the network performed a task. This security property is very relevant in clinical environments, where be able to prove that a drug has been taken by a patient.

- **Privacy and Anonymity:** These are very important in scenarios where the location and identities of the nodes should be hidden or protected. For example, in clinical environments where a patient's node represent to its patient, therefore it transcends beyond the technological dimension and affect its social environment, since sensor networks are used as a surveillance tool to collect data about the patient's vital sign. This feature is so important in clinical environments, that our ongoing work is focused on include an Identification Management Platform (IdM) to ensure privacy and anonymity of patients.

III. Overview of Security Mechanism

WSNs are highly constrained from different aspects. These constrains need to be taken into account in order to protect WSNs to the attacks and threats presented in Section II. Remark that these security requirements and the WSN vulnerabilities are increased with the apparition of the Internet of things, which allows the connection of WSNs to the Internet.

Our research is focused in cryptographic primitives, which are required by the security protocols and mechanisms in order to integrate the security properties into their operations. These cryptographic primitives are Symmetric Key Cryptography (SKC), Public Key Cryptography (PKC), and Hash Functions.

A. Symmetric Key Cryptography (SKC) and Hash Functions

Symmetric Key Cryptography (SKC) provides confidentiality and integrity to the communication channel, and requires 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. The majority of WSNs are based on the IEEE 802.15.4 standard, which offers three levels of security: Hash Functions, Symmetric Key Cryptography and both [8], [17].

B. Public Key Cryptography (PKC)

Public Key Cryptography (PKC), also known as asymmetric cryptography, is useful for secure broadcasting and authentication purposes. It requires of 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. Hash Functions are used to create "digital fingerprints" of data. This property can be used to build other cryptographic primitives like the mentioned Message Authentication Code (MAC), which provides authenticity and integrity in the messages. 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.

Public Key Cryptography was considered unsuitable for sensor node platforms, but that assumption was a long time ago. The approach that made PKC possible and usable in sensor nodes was Elliptic Curve Cryptography (ECC), which is based on the algebraic structure of elliptic curves over finite fields. Some studies has been carried out about RSA in reduce chips [25], but it was non-viable, this study is going also to present optimizations for RSA, but mainly is going to be focused on ECC, since it has smaller requirements both in computation and memory storage, due to its small key sizes and its simpler primitives [26].

The related works of the implementation of an efficient cryptographic algorithm for constrained devices have been focus on the optimizations of the multiplication operation, since it is the most expensive operation in RSA and ECC algorithms. There several implementations of RSA and ECC, ECC can be over either $GF(2^m)$ or $GF(p)$, we have focused on modular arithmetic since this can be applied for RSA and ECC, and this has similar nature to the arithmetic of the micro-controller used in the 6LoWPAN devices. However, some interesting approaches have been defined for ECC implementations over $GF(2^m)$, for example [28] has showed that field multiplication is faster over $GF(2^m)$ than in $GF(p)$ optimizing multiplications for $GF(p)$, in order to define an arithmetic of the micro-controller closer to $GF(2^m)$ has been suggested some new hardware implementations [29]. Some of the most known software implementations over modular arithmetic are, on one hand, for RSA with assembly code and instruction set extension on a 8-bit ATmega128 [25], They presents a hybrid multiplication algorithm exploiting advantages of operand and product scanning multiplication algorithm to reduce the number of memory accesses. On other hand, for ECC are TinyECC [22], [23] and NanoECC [24], which implement ECC-based signature generation and verification (ECDSA), encryption and decryption (ECIES), and key agreement (ECDH). TinyECC adopted several optimization techniques such as optimized modular reduction using pseudo-Mersenne prime, sliding window method, Jacobian coordinate systems, inline assembly and hybrid multiplication to achieve computational efficiency. Note that the computational and memory requirements of these algorithms are not small (e.g. ECDSA requires 19308 bytes ROM and 1510 bytes RAM for the MICAz, generating a signature in 2 seconds. and verifying it in 2.43 seconds), although the implementation of these primitives is constantly evolving and improving, one of that news approaches to this challenge is the work presented in this paper.

The same idea of hardware focused implementations [29] and exploit specific hardware supported instructions [25] in order to improve and simplify the multiplication is what we are presenting in this paper. Specifically, this work proposes an implementation of multiplication operation for RSA and ECC based on bit shifting and additions instead of the microprocessor's multiplication operation. This approaches results interesting for microprocessor, which has not hardware support for multiplication operation such as MSP430, where it is actually emulated and consequently requires a big amount of clock cycles, while the bit shifting is supported and it just needs between 1 and 4 cycles (depends on data is in a register or in memory).

IV. Mathematical Optimization based on bit shifting instead of microprocessor's multiplication operation

There is an important literature about the advantages of the Montgomery's representation for this calculation, see V-A1. For that reason, it is used in our solution for both solutions i.e. based on bit shifting and multiplication operation. RSA and ECC require two different integer sizes (k). Specifically, it is considered 1024-bit for RSA, i.e. $k = 1024, R = 2^{1024}$, and 160-bit for ECC, i.e. $k = 160, R = 2^{160}$).

In Montgomery representation for representing the numbers a and b, which are going to be multiplied, we have aR and bR mod n (n is a prime for ECC, and the multiplication of two primes in RSA). Addition and subtraction operations with these numbers do not cause problems since R is common factor. The problem is coming with the multiplication operation, when aR and bR is multiplied, we get abR^2, but what we need is abR. Therefore, we have to reduce it by a factor R. The great advantage of Montgomery representation is just to avoid that, since Montgomery offers the reduction of the factor R during multiplication; reaching, in this way, a more effective multiplication.

Such as mentioned, the multiplication operation is what consumes the higher part of the time, since it is repeated thousands of times. For that reason, multiplication operation is what we will go to optimize and discuss more in detail in the next subsections, specifically we are going to define two ways to carry it out: bit shifting and microprocessor's multiplication operation.

A. Bit shifting

Let a and b two integers in Montgomery representation. Then, such as mentioned, we have aR and bR mod n that assume between 0 and $n-1$. They are stored in binary representation, so that $aR = \sum_i a_i 2^i$ and $bR = \sum_i b_i 2^i$.

We calculate $(aR)(bR)R^{-1} = (ab)R$, therefore we need to carry out k right bit shiftings (with $k = 160$ or 1024 according to the case.)

Recall that modulus n is odd (either it is a prime in ECC, or it is the multiplication of two primes in RSA), thus when it is divided by 2 mod n, two options are defined: either it is even number and it can be directly shifted, or it is odd number and consequently needs to add n, in order to reach 0 in the least significant bit and be able to shift it.

For the multiplication process is needed a variable to accumulate the current result, we will call to that variable P, which digits are $P = \sum_i P_i 2^i$. Each one of the digits B_i is going to be multiplied by $\sum_i A_i 2^i$ and divided by 2. As initially P is 0, if $B_i = 0$ for some initial values, it can be ignored. Therefore, this starts directly by the digit in the position i_0, such that $B_{i_0} = 1$, and copy the value of A_i in P_i.

From the position i_0, we can find in the next steps: $B_i = 0$ or 1. On one hand, when $B_i = 0$, it has to divide P by 2, and add n when P is odd. On other hand, when $B_i = 1$, then it need to add the value of aR to P, before it is divided by 2.

To estimate the time, we will consider that the probability of finding $B_i = 1$ or 0 is the same, i.e. $1/2$. Therefore, for each k bits of B_i, when it is 1, it needs to carry out an addition of k bits (i.e. addition of a_i) and a division by 2 of P. And when it is 0 just only one right bit shifting. Therefore, k divisions by 2 and $k/2$ additions. Since, each division by 2 is, such as mentioned, when it is a shifting and 1/2 times also an addition of n. The total time is:

$k(d + s/2) + (k/2)s = k(d + s)$, where d is the time for k right bit shiftings, and s is the time for k bits addition.

Since the microprocessor MSP430 offers 16-bits operations, additions and bits shifting are defined in blocks of 16-bits. Therefore, we call α to the time for 16-bits additions and shifting (usually 1-4 CPU cycle for bit shifting and 1-6 cycles for additions, depends on access to memory and registers). The final time is:

$$2\alpha k^2/16 = \alpha k^2/8.$$

The program code of the bit shifting algorithm explained is presented in the Program 1, but since this is critical part of the program, we have programmed it in assembler code, see Program 3.

Program 1 Code based on Bit shifting

```
for(i=0;i<k;i++)
   P[i]=0;

for(i=0;i<k-1;i++){
   for(shift = 0x01; shift!=0; shift<<=1){
     if(b[i] & shift) add(P,a);
     if(P[0] & 0x01) add(P,n);

     for(j=0;j<k-1;j++){ /*Right bit shifting*/
       P[j] >>= 1;
       if(P[j+1] & 0x01) P[j] |= 0x80;
     }
     P[k-1] >>= 1;
   }
}

if(isGreaterEqual(P,n))
   sub(P,n);
```

B. Microprocessor's multiplication operation

Let an instruction from the microprocessor's set of instructions to carry out multiplication operation, which operated 2 registers of 16 bits and save the 32 bits of the result in two registers of 16 bits. For example, this instruction is simulated in MSP430 chip, which is the microprocessor used by Tmote Sky for our evaluation, Program 2 presents the internal operations to support 16-bits x 16-bits multiplication. We are going to call μ to the time spent by that operation, and α for the time of 16-bits additions and 16-bits shifting.

Let the next numbers to apply the multiplication $aR = \sum_j \overline{A}_j 2^{16j}$ and $bR = \sum_j \overline{B}_j 2^{16j}$. In this case j values are between 0 and $k/16$, instead of between 0 and k from Program 1. Therefore, for each multiplication of k bits, we need to carry out $k/16$ 16-bits multiplications and $2k/16$ additions, getting like result a number equal to $k + 16$ bits. Therefore, the time is equal to: $(\mu + 2\alpha)k/16$.

For each step of the multiplication of aR by the digits of $\overline{B}_i$, since multiplication is carried out in blocks of 16 bits, this

needs to add the result with the previous result i.e. an addition of (a sum of two numbers k bits and 16 bits, so $\alpha(k+1)/16$ additions), then this needs to divide it by 2^{16} mod n, we call δ for the time used for the division by 2^{16}.

The total time for each one of the 16 bits blocks ($k/16$ blocks, $\overline{B}_i$) is $(\mu+2\alpha)k/16+\alpha(k+1)/16 = \frac{\mu k+\alpha(3k+1)}{16}$, and addition δ, i.e. the total time is $\frac{\mu k^2+\alpha(3k+1)k}{256}+\frac{k\delta}{16}$.

Division of a number of $k+16$ bits by 2^{16} mod n is carried out adding (or subtracting) n until that the result is multiple of 2^{16}. If the last digit of n in base 2^{16} is equals to 1 the process is simple, since the number of times to subtract n is indicated by the last digit of the number that we want to divide by 2^{16} mod n. Therefore the total time is equal to:

$$\delta_t = \frac{(\mu+3\alpha)k+\alpha}{16}.$$

δ_t is the ideal time, when we have chosen n such that its last digit is equal to 1, in order to carry out in a simple way the division. In a general case we cannot assume the value of the last digit of n is equal to 1, therefore this estimation is not realistic. But, Extended Euclidean algorithm can be used, in order to fix the process pre-calculating the modular multiplicative inverse of the last digit of n mod 2^{16} and it can be used with the last digit of p. Therefore we reach the next time, which is more realistic.

$$\delta = \frac{k}{16}(\mu+3\alpha)+2\mu+2\alpha.$$

This can be simplified by assuming that terms that do not have k are not so relevant for the total time. Therefore δ_t and δ are quite similar, in the order of $\frac{k}{16}(\mu+3\alpha)$. Therefore, based on that expression and reducing terms that do not have k^2, the next total time for microprocessor's multiplication operation is defined:

$$M \simeq \frac{\mu k^2}{256}+\frac{(\mu+3\alpha)k^2}{256} = \frac{(2\mu+3\alpha)k^2}{256}.$$

Program 2 Code emulated for Multiplication in MSP430 [30]

```
; EXECUTION TIMES FOR REGISTERS CONTENTS (CYCLES) without CALL:
; MPYU: Unsigned 16 x 16-bit Multiplication
; MACU: Multiplication and Accumulation
; TASK       MACU MPYU            EXAMPLE
; MINIMUM     132 134   00000h x 00000h = 000000000h
; MEDIUM      148 150   0A5A5h x 05A5Ah = 03A763E02h
; MAXIMUM     164 166   0FFFFh x 0FFFFh = 0FFFE0001h
; UNSIGNED MULTIPLY SUBROUTINE: IROP1 x IROP2L > IRACM/IRACL
;
; USED REGISTERS IROP1, IROP2L, IROP2M, IRACL, IRACM, IRBT
;
MPYU CLR IRACL ; 0 > LSBs RESULT
CLR IRACM ; 0 > MSBs RESULT
; UNSIGNED MULTIPLY AND ACCUMULATE SUBROUTINE:
; (IROP1 x IROP2L) + IRACM|IRACL > IRACM|IRACL
;
MACU CLR IROP2M ; MSBs MULTIPLIER
MOV #1,IRBT ; BIT TEST REGISTER
L$002  BIT IRBT,IROP1 ; TEST ACTUAL BIT
JZ L$01 ; IF 0: DO NOTHING
ADD IROP2L,IRACL ; IF 1: ADD MULTIPLIER TO RESULT
ADDC IROP2M,IRACM
L$01  RLA IROP2L ; MULTIPLIER x 2
RLC IROP2M ;
;
RLA IRBT ; NEXT BIT TO TEST
JNC L$002 ; IF BIT IN CARRY: FINISHED
RET
```

C. Comparative between bit shifting and microprocessor's multiplication operation

The comparative between bit shifting and microprocessor's multiplication operation shows us that in a general way bit shifting is better than microprocessor's multiplication operation, when the following equation is true:

$$\frac{\alpha k^2}{8} < \frac{(2\mu+3\alpha)k^2}{256} \Rightarrow 32\alpha < 2\mu+3\alpha \Rightarrow \frac{29}{2} < \frac{\mu}{\alpha}.$$

In conclusion, when the number of cycles to carry out microprocessor's multiplication operation is more than 15 times the cycles to carry out addition or bit shifting, it is preferable bit shifting solution. Since, MSP430 microprocessor's multiplication operation requires a big amount of clock cycles (150 cycles in MSP430), while the bit shifting and additions are supported and it just needs between 1 n 4 cycles for bit shifting and 1 and 6 cycles for addition, this depends on the access to registers and memory, i.e. $rrcR4$, i.e. bit shifting for registers is just 1 cycle, but $rrc0(R1)$, which is bit shifting in the memory address with value $R1$ are 4 cycles, at the same way for add operation. Therefore, the evaluation has presented that bit shifting is better than microprocessor's multiplication operation with a relation of when its cost is 15 times or less than multiplication i,e, $\mu < 15\alpha$, and MSP430 has a μ/α between 38 and 150.

In addition, other optimizations have been considered, which are presented in the Section V. Finally, a real evaluation over Tmote Sky, i.e. MSP430 microprocessor, is carried out in Section VI.

V. Other Mathematical Optimizations of Security Mechanism based on PKC

In addition to the optimization presented in the previous section. we are going to define additional optimizations, which have been highly mentioned and used in the related works.

In both cases, ECC and RSA, we have an Abelian group G with an operation. In ECC it is used the group of rational points in a elliptic curve with additive notation and in RSA the multiplicative group $\mathbb{Z}_n^*$.

The operation in G involves addition, multiplication (for RSA) and also modular inverses (for ECC). Modular inverses can be computed using several multiplications and additions. The most critical part is the integer multiplication, since this is the most expensive operation. Optimizations made in this part of the problem will be called *Optimizations on Multiplication*, such as the already mentioned in the Section IV.

Abelian groups are $\mathbb{Z}$-modulus over the ring of integers $\mathbb{Z}$, therefore we have an operation $G \times \mathbb{Z} \rightarrow G$. Given $g \in G$ and $e \in \mathbb{Z}$ we need to compute $g \cdot e$ (or g^e if we are using multiplicative notation). In both cases e use to be a big integer. The optimizations made in this part will be called *Optimizations on Powering*.

There is a third kind of optimizations that have been considered, they are the optimizations related to the way we represent the integers. The usual binary format is not the best in most of the cases. These optimizations will be considered in *Integer Representations*.

A. Integer Representations

We have considered the usual binary representation for integers, and also the following ones:

1) Montgomery Representation for the bases: Let n be a positive integer and consider the ring of modular integers $\mathbb{Z}_n$. Two integers a and b are equivalent in $\mathbb{Z}_n$ if $a - b = nt$ for some $t \in \mathbb{Z}$. There are exactly n different classes in $\mathbb{Z}$, the elements equivalent to $0, 1, \cdots, n-1$. When we make sums and multiplications over the elements in $\mathbb{Z}_n$ and the result is out of the range $0, \cdots, n-1$ we use to compute the element inside this range that it is equivalent to our result. This operation requires a division by n, that is rather time consuming. The Montgomery representation is probably the best one for modular arithmetic. It is very well known and requires a not very high computational effort.

It is based on the fact that we can use a multiplicative unit R in $\mathbb{Z}_n$ and pre-compute the modular inverse R^{-1}. In order to represent an element $a \in \mathbb{Z}_n$, we keep the product $aR(modn)$. The multiplication by R is of course a bijection over the set $\mathbb{Z}_n$ (the inverse operation is to multiply by R^{-1}), therefore we have the same information. But with this representation it is easier to keep the sums and the products inside the range $0, \cdots, n-1$.

This system is useful when we have to make a lot of operations in modular arithmetic (this is our case) and we do not need to make Montgomery reductions (going from aR to a and backwards). This representation is rather usual and the details can be found in [5].

2) NAF Representation for the exponents: The NAF (non-adjacent form) representation uses the digits 0, 1 and -1 to represent an integer. An integer is in NAF representation if non-zero digits are not adjacent. For example, the number $(1, 0, 0, -1)$ will be $1 \cdot 2^3 + (-1) \cdot 2^0 = 8 - 1 = 7$. This representation reduces the number of multiplications when we compute g^e (or $g \cdot e$ in additive notation). This representation is well known in cryptography. We have also used some tricks based on the fact that two non-zero digits cannot be adjacent in order to encode the there possible values $0, 1$ and -1 in binary format with minimal memory consume. This last optimization will be called compacted NAF representation.

B. Optimizations on Powering

The following algorithms are used to compute g^e (or $g \cdot e$ in additive notation) when e is written in binary notation $e = \sum_{i=0}^{k} e_i 2^i$. The binary digits e_i are used from 0 to k in the right to left algorithm and from k to 0 in the left to right one.

Right to Left Powering

$b = g$
$result = 1$ (neutral element in G)
for $i = 0$ to k **do**
  if e_i equals 1 **then**
    $result = groupOperation(result, b)$
  end if
  $b = groupOperation(b, b)$
end for

Left to Right Powering

$result = g$
for $i = k - 1$ to 0 **do**
  $result = groupOperation(result, result)$
  if e_i equals 1 **then**
    $result = groupOperation(result, g)$
  end if
end for

Both algorithms are rather similar, but the R-L one needs one more variable. Remark, that both algorithms can be easily integrated with NAF representation.

C. Optimizations on Multiplication

We have also considered the next optimizations, but finally we have applied the solution based on Montgomery with NAF and L-R powering, such as presented in Table II. This optimizations has been carried out with bit shifting such as mentioned in Section IV.

1) Booth: Booth's multiplication algorithm is a rather faster alternative to multiply integers. This algorithm check two bits of the operands and perform different operations on each alternative $00, 01, 10, 11$. After that operations the variable is shifted. Mathematical details of the algorithm can be found in different sources, for example [6]. We have also considered a faster version of the algorithm in which four bits are used.

2) Karatsuba-Ofman: The details of this multiplication algorithm can be found in [4, Section 4.3.3]. The basic idea of this algorithm is as follows: x and y are $2k$-bits operands, and suppose x and y are $x = x_1 2^k + x_0$ and $y = y_1 2^k + y_0$. Then we can multiply x and y making the following half-size operations $x_1 y_1$, $x_0 y_1 + x_1 y_0$, $x_0 y_0$.

We can apply this idea again to the products $x_i y_j$ and reduce again the product to half-size operands. This recursive approach needs a lot of computational resources and it is not so effective. We have applied the method once and the results suggest that the method is not effective in our situation.

3) Barrett: We have mentioned previously that the sums and multiplications had to be made in $\mathbb{Z}_n$, therefore, after making the product xy we have to compute the equivalent result inside the range $0, \cdot, n-1$. The Barrett algorithm is used to make this reduction. The mathematical details can be found in [1].

D. Modular Inverses

Given $x \in \mathbb{Z}_n$, the modular inverse of x is the element $y \in \mathbb{Z}_n$ such that xy is equivalent to 1 in $\mathbb{Z}_n$. This element y exists and it is unique when $\gcd(x, n) = 1$. We do not need to compute modular inverses for RSA, but ECC requires computation of them. We have considered two different algorithms for that:

1) Powering: Having in mind that $x^{\varphi(n)} = 1$ in $\mathbb{Z}_n$ (Euler formula), we know that $x^{\varphi(n)-1} x = 1$, therefore $x^{\varphi(n)-1}$ is the (unique) modular inverse of x. In ECC we compute inverses in $\mathbb{Z}_p$ with p prime, therefore we know that $\varphi(p) = p - 1$ and this operation involves just powering x. This algorithm will have $O(\log^3(p))$.

2) Extended Euclid Algorithm: If $\gcd(x, n) = 1$, then we can find u and v such that $xu+yv = 1$. These elements u and v can be computed by Euclid's Extended Algorithm. We have used the binary version given in [3, Algorithm 1.3.8] with the appropriate reductions.

Both algorithms have been compared in [2]. The first one it a bit slower, but requires less variables than the second one. Finally, we have applied for our solution Extended Euclid Algorithm with some adoptions, in order to support Montgomery representation.

VI. Results and Evaluation

The evaluation of the algorithms optimized has been initially evaluated over the Intel 8051 Simulator Keil from ARM, and the results have been verified with the cryptographic library LiDIA [18]. Finally, this has been evaluated over real motes, specifically over Tmote Sky with the Contiki 2.4 OS [19].

The features and hardware resources of the Tmote Sky are:

- Wireless Transceiver Chipcon CC2420 (includes Intel 8051, 8 bits CPU), based on IEEE 802.15.4 (2.4 GHz)
- Bandwidth of 250 Kbps
- Microcontroller Texas Instrument MSP430 F1611 of 16 bits and 8 MHz (RAM: 10 KB and Flash: 48 KB)
- ADC, DAC, and DMA
- Humidity, temperature and lighting sensors. 16 ports to support external I/O devices and sensors.
- Low energy consumption. Quick wake-up *sleep* (<6 μs)
- Hardware support for Symmetric Key Cryptographic, specifically AES until 128 bits key length.

A. Results for ECC and RSA algorithms in Tmote Sky

The results from the different mathematical optimizations presented in the Section V for RSA are presented in a ranking of the different algorithms with respect to memory and speed are presented in the Table I. At the same way, the results for ECC are presented in a ranking of the different algorithms with respect to memory and speed is presented in the Table II.

TABLE I
RSA results: Left column presents algorithms from the fastest (top) to the slowest (bottom). Right column presents algorithms from the least weight (top) to the most (bottom)

	Montgomery + NAF L-R	Classic L-R	
↑	Montg. + Booth Modif. L-R	Classic R-L	M
S	Montgomery L-R	Montgomery L-R	E
P	NAF L-R	Booth L-R	M
E	Booth Modif. L-R	Booth R-L	O
E	NAF Compact. L-R	Booth Modif L-R	R
D	Classic L-R	NAF Compact L-R	Y
	Booth L-R	Montg. + Booth Modif. L-R	↓
	Classic R-L	NAF L-R	
	Booth R-L	Montgomery + NAF L-R	

TABLE II
ECC results: Left column presents algorithms from the fastest (top) to the slowest (bottom). Right column presents algorithms from the least weight (top) to the most (bottom)

	Montgomery + NAF L-R	Classic L-R	
↑	Montg. + Booth Modif. L-R	Classic R-L	M
S	Montgomery L-R	Booth L-R	E
P	Montgomery R-L	Booth Modif. L-R	M
E	NAF Compact. L-R	NAF Compact. L-R	O
E	Booth Modif. L-R	Montgomery L-R	R
D	Montgomery-Kaliski R-L	Montgomery R-L	Y
	Classic L-R	Montgomery-Kaliski R-L	↓
	Booth L-R	Montg. + Booth Modif. L-R	
	Classic R-L	Montgomery + NAF L-R	

There is one particular aspect of sensor network security that is commonly neglected or overlooked: the relationship between the security requirements, the features of the application and its context, and the security mechanisms. Indeed, the context and the requirements of a specific application have a great influence on the security mechanisms that should be used to protect the network [20]. For that reason, the results are presented as a balance between time and memory requirements, since depends on our application requirements and devices, we are able to choose a solution where speed is optimized or another where memory usage is optimized.

The quickest ECC algorithm is based on *Montgomery + NAF L-R*, such as presented in Table II, this has been optimized for MSP430 with bit shifting in assembler language for the Montgomery multiplication for 160 bits. We have optimized Montgomery multiplication, which is carried out in 12480 cycles (1,5625 milliseconds in the 8 Mhz MSP430 microprocessor of Tmote Sky). In order to reach this solution, we have used 10 microprocessor's registers to keep the 160 bits variable with the partial multiplication results, with this optimization we have reduced almost the 40% of the total number of cycles, since rrc operation for bit shifting, and add operation for addition spend 1 cycle and 3 cycles respectively, instead of 4 and 6. The Program 3 shows as bit shifting of the 160 bits is carried out in just 10 cycles, and addition in 30 cycles with this optimization. Finally, we have unrolled loops in order to optimize more the final assembler code.

VII. Conclusion

Internet of things, and particularly 6LoWPAN is defining a new challenge for security, since wireless sensor networks are being connected to the Internet. Therefore, it is necessary to provide efficient and usable security mechanisms that could protect the WSN against attacks. While it was possible to deploy a secure sensor network based on AES for certain applications, there are some new challenges that need to be considered with this extension to the internet. IoT requires a new scalability level, which can not be satisfied with Symmetric Key Cryptography (SKC). For that reason, it is required Public Key Cryptography (PKC), it is evaluated and optimized RSA and ECC algorithms. Finally, it has been evaluated over MSP430 microprocessor from Tmote Sky.

This evaluation has concluded that ECC is a suitable solution for Future Internet devices, such as 6LoWPAN

Program 3 Sketch of assembler code for Montgomery's multiplication based on bit shifting

```
//res, where is stored the partial result, is kept in the register since %0 to %9
" mov.w #0,%0 \n\t mov.w #0,%1 \n\t mov.w #0,%2 \n\t mov.w #0,%3 \n\t mov.w #0,%4
mov.w #0,%5 \n\t mov.w #0,%6 \n\t mov.w #0,%7 \n\t mov.w #0,%8 \n\t mov.w #0,%9 \n\t"

//keep in the memory stack the fist operand and the second one divided by 2
" rrc.w 18(%10) \n\t push.w 18(%10) \n\t"

//The same between 18 and 0 in block of 2 bytes (16 bits)
" rrc.w 0(%10) \n\t push.w 0(%10) \n\t"

//We carry out two versions depending on the carry flag (oddity of the second operand).
//Depending on the bits of the first operand we have to repeat several additions
//and bit shifting of the registers where is stored the partial result.

//ADDITION OF 160 BITS IS CARRIED OUT IN 30 CYCLES
".LsA2: add.w 2(r1),%0 \n\t"
" addc.w 4(r1),%1 \n\t"
" addc.w 6(r1),%2 \n\t"
" addc.w 8(r1),%3 \n\t"
" addc.w 10(r1),%4 \n\t"
" addc.w 12(r1),%5 \n\t"
" addc.w 14(r1),%6 \n\t"
" addc.w 16(r1),%7 \n\t"
" addc.w 18(r1),%8 \n\t"
" addc.w 20(r1),%9 \n\t"

//BIT SHIFTING OF 160 BITS IS CARRIED OUT IN 10 CYCLES
" rrc %9 \n\t rrc %8 \n\t rrc %7 \n\t rrc %6 \n\t rrc %5 \n\t rrc %4 \n\t rrc %3 \n\t
rrc %2 \n\t rrc %1 \n\t rrc %0 \n\t"
```

nodes, since time spent for Montgomery multiplication is just 1,5625 milliseconds.

Ongoing work is focused on assembler implementation of the modular inverses, in order to define a full optimized exponentiation to implement ECDSA (for digital signature), ECIES (for data encryption), and ECDH (for key establishment). In order to compare our solution with respect to other current implementations such as TinyECC, which is the best public implementation of ECC for these kind of devices.

ACKNOWLEDGMENT

This work has been carried out in frames of, on one hand, the grants from the Fundación Séneca "Programa de Ayuda a los Grupos de Excelencia 04552/GERM/06" and 12006/PI/09, and on other hand, the project from the Ministry of Science and Innovation of Spain (MTM2009-11696). Finally, the authors would like to thank the University of Murcia by the scholarship: *Becas/Contrato Predoctorales*.

REFERENCES

[1] P.D. Barrett, *Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor*, Advances in Cryptology. LNCS, vol. 263, pp 311-323. Springer, 1987.

[2] R.P. Brent, *Some integer factorization algorithms using elliptic curves*, in Proc. 9th Australian Computer Science Conference, 1985.

[3] H. Cohen, *A Course in Computational Algebraic Number Theory*, 3rd ed. GTM 138, Springer 1996.

[4] D. Knuth, *The Art of Computer Programming 2. Seminumerical Algorithms*, 3rd ed. Addison-Wesley, 1997.

[5] P. Montgomery, *Modular Multiplication Without Trial Division*, Math. Computation, vol. 44, pp. 519–521, 1985.

[6] W. Stallings. *Computer Organization and Architecture: Designing for performance*, 5th ed. ISBN 0-13-081294-3. New Jersey: Prentice-Hall, Inc. 2000.

[7] A. Sleman, and R. Moeller. *Integration of Wireless Sensor Network Services into other Home and Industrial networks; using Device Profile for Web Services (DPWS)*, Information and Communication Technologies: From Theory to Applications 2008, pp. 1-5, 2008.

[8] 802.15.4-2003, IEEE Standard, *Wireless medium access control and physical layer specifications for low-rate wireless personal area networks*, May 2003.

[9] J.W. Hui, and D.E. Culler, *Extending IP to Low-Power, Wireless Personal Area Networks*, IEEE Internet Computing, Vol. 12, Iss. 4, pp.37-45, 2008.

[10] N. Kushalnagar, G. Montenegro, J. Hui, and D. Culler, *6LoWPAN: Transmission of IPv6 Packets over IEEE 802.15.4 Networks*, RFC 4944, September 2007.

[11] Xin Ma and Wei Luo, *The Analysis of 6LowPAN Technology*, Computational Intelligence and Industrial Application. PACIIA '08. Volume 1, pp. 963-966, 2008,

[12] A.J. Jara, R.M. Silva, J. Sa Silva, M.A. Zamora, and A.F.G. Skarmeta, *Mobile IPv6 over Wireless Sensor Networks (6LoWPAN) Issues and feasibility*. European Wireless Sensor Networks, 2010.

[13] A.J. Jara, A.J., M.A. Zamora, and A.F.G. Skarmeta,*HWSN6: Hospital Wireless Sensor Networks based on 6LoWPAN Technology: Mobility Fault Tolerance Management*. IEEE International Conference on Computational Science and Engineering, 2009.

[14] J. Granjal, R. Silva, E. Monteiro, J. Sa Silva, and F. Boavida. *Why is IPSec a viable option for wireless sensor networks*, Wireless and Sensor Networks Security, 2008.

[15] H. Mukhtar, Kim Kang-Myo, S.A. Chaudhry, A.H. Akbar, Kim Ki-H, and S-W Y., *LNMP-Management architecture for 6LoWPAN*, Network Operations and Management Symposium, 2008.

[16] Z. Shelby, P. Thurbert, J. Hui, and S. Chakrabarti, C. Bormann and E. Nordmark, *6LoWPAN Neighbor Discovery*, draft-ietf-6lowpan-nd-10, Internet-Draft IETF, work in progress, 2010.

[17] Carolina Tripp Barba. *Impacto de mecanismos de seguridad en el funcionamiento de sensores IEEE 802.15.4*, *In Spanish*, 2008.

[18] S. Hamdy, *LiDIA. A library for computational number theory. Reference Manual*. Edition 2.1.1, 2004.

[19] A. Dunkels, O. sterlind. *Contiki Programming Course: Hands-On Session Notes*. Swedish Institute of Computer Science, 2008.

[20] J. Lopez, R. Roman, and C. Alcaraz. *Analysis of Security Threats, Requirements, Technologies and Standards in Wireless Sensor Networks*, In Foundations of Security Analysis and Design V: FOSAD 2007/2008/2009 Tutorial Lectures, LNCS, vol. 5705. Springer-Verlag, 289-338. 2009.

[21] R. Roman, J. Lopez. *Integrating Wireless Sensor Networks and the Internet: A Security Analysis*. Internet Research 19(2), 246-259 (2009)

[22] A. Liu, P. Ning. *TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks.* 7th International Conference on Information Processing in Sensor Networks, SPOTS Track, USA, pp. 245-256, 2008.

[23] S.C. Seo, D.-G. Han, H.C. Kim, S. Hong, *TinyECCK: Efficient Elliptic Curve Cryptography Implementation over GF(2m) on 8-bit MICAz Mote.* IEICE Transactions on Info and Systems E91-D(5), 1338-1347, 2008.

[24] P. Szczechowiak, L.B Oliveira, M. Scott, M. Collier, R. Dahab. *NanoECC: Testing the Limits of Elliptic Curve Crytography in Sensor Networks.* Dublin City University, Ireland; UNICAMP, Brasil, 2008.

[25] N. Gura, A. Patel, A. Wander, H. Eberle, S.C. Shantz. *Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs*. Workshop on Cryptographic Hardware and Embedded Systems, 2004.

[26] Y. Hitchcock, E. Dawson, A. Clark, P. Montague. *Implementing an efficient elliptic curve cryptosystem over GF(p) on a smart card*. ANZIAM Journal, 2003.

[27] J. Ramio Aguirre. *Seguridad Informatica y Criptografia. In Spanish.* Universidad Politecnica, Departamento de Publicaciones, 2006.

[28] Leif Uhsadel, Axel Poschmann, and Christof Paar. *Enabling Full-Size Public-Key Algorithms on 8-bit Sensor Nodes*. European Workshop on Security and Privacy in Ad hoc and Sensor Networks, 2007.

[29] A. Hodjat, L. Batina, D. Hwang, I. Verbauwhede, *HW/SW Co-Design of a Hyperelliptic Curve Cryptosystem using a Microcode Instruction Set Coprocessor* Integration, VLSI Journal 40(1), pp.45-51, 2007.

[30] Lutz Bierl, MSP430 Family Mixed-Signal Microcontroller Application Reports, http://focus.ti.com.cn/cn/lit/an/slaa024/slaa024.pdf, pp. 478-480, 2000.