An Estimator for the ASIC Footprint Area of Lightweight Cryptographic Algorithms

Honorio Martín, Pedro Peris-Lopez, Juan E. Tapiador and Enrique San Millán

***Abstract*—In resource-constrained devices such as RFID tags or implantable medical devices, algorithm designers need to make careful choices to ensure that their proposals are sufficiently efficient for the target platform. A common way of expressing such restrictions is in terms of an upper bound for the maximum available footprint area in gate equivalents (GE). For example, RFID tags conforming to standards EPC Class-1 Generation-2 and ISO/IEC 18000-6C can devote up to 4K GE to security functions. However, in most cases algorithm designers are not hardware experts, nor they have any quantitative means to find out how much area their designs would occupy in a given technology. In this paper, we attempt to fill this gap by providing an estimate of the upper bound for the footprint area of any algorithm. Our approach takes into account the main components of such algorithms, namely basic arithmetic/logic operations and additional hardware such as registers and multiplexers. We believe that our proposal can help designers in making informed decisions about what kind of algorithmic structures can be afforded for a target environment.**

***Index Terms*—ASIC implementations, Footprint area, RFID, VHDL, Lightweight algorithms.**

I. Introduction

RADIO Frequency Identification (RFID) technology, which uses radio signals for automatic identification, has experienced a notable growth in the last years [1], [2], [3]. An RFID tag responds to a reader request by using the radio channel and including in its answer a value that allows its unequivocal identification. This value is generated by the tag and should be protected to guarantee the privacy of the tag holder. The security of RFID systems is a challenging problem, particularly for low-cost devices such as those covered by the EPC Class-1 Generation-2 [4] and ISO/IEC 18000-6C [5] standards. This is usually achieved by incorporating some cryptographic modules [6], [7], [8] upon which security services can be provided through different protocols (see, e.g., [9], [10], [11], [12]).

A major difficulty in providing RFID tags with security functions comes from the scarcity of computational resources available in such platforms (see, e.g. [13], [14], [15], [16] for recent developments in the design of tiny tags). For example, in a low-cost RFID tag the Gate Equivalents (GE) that can be allocated to security issues have to be in the range between 250 and 4K GE, which restricts affordable solutions to lightweight algorithms only.

H. Martin and E. San Millan are with the Department of Electronic Technology, Carlos III University of Madrid, Spain e-mail: {hmartin,quique}@ing.uc3m.es

P. Peris-Lopez and J. E. Tapiador are with the Computer Secuirty Lab, Carlos III University of Madrid, Spain e-mail: {pperis,jestevez}@inf.uc3m.es

Proposals in the area of lightweight cryptography have proliferated over the last years, but most of them do not provide results regarding their implementation. In many cases, arguments in favor of their lightweightness are based on the use of some operations that are generally considered inexpensive. However, this assumption is not always true, and the implementation of some proposals have shown that the area limit of 4K GE is often surpassed. In other cases, the design turns out to be not so lightweight because of factors such as the bit length of the variables, the need for additional memory blocks –which is usually missed in the analysis of resources– , and the overhead imposed by selection and control logic. These and other aspects often make the final gate count much higher than expected.

All in all, providing accurate estimates of the footprint area of an ASIC implementation is a hard task for algorithm designers [17], [18], [19]. More often than not, designers do not have the hardware design skills nor the tools required to implement and analyze their proposals. Furthermore, there is a lack of a standard methodology to provide such an estimation, as the result will vary depending on factors such as the chosen architecture, the manufacturing technology, the possibility of introducing optimizations at various levels, etc. Problems such as this are common in other related areas. For example, designing low-power embedded systems [20], [21], [22] is also a major problem nowadays, and system designers face a situation similar to that described above. Very recently, Ben Atitallah *et al.* [23] have presented a methodology to provide designers with an estimation of the power consumption of a complete system. Similarly, in this paper we propose a relatively simple estimator for the footprint area occupied by the ASIC implementation of an algorithm. The suggested formula requires the designer to know only a few high-level details about the target implementation, such as the number of registers used to store inputs, outputs and intermediate variables, and some parameters related to the control structures. In principle, our work is motivated by and focussed on lightweight cryptographic algorithms for constrained platforms such as RFID tags or implantable medical devices. Despite this, we believe our approach could be easily extended to non-security designs by incorporating additional factors into our methodology.

The rest of this paper is organized as follows. In Section II we provide an overview of the main basic operations that can be used in lightweight algorithms, and in Section III we show the hardware architectures employed to implement them. Section IV discusses our results synthesizing basic operation blocks using two real manufacturing libraries. Subsequently, in

Section V we present a method to estimate the footprint area of a whole algorithm and discuss our experimental results with a battery of real-world examples. Finally, Section VI concludes the paper and summarizes our main contributions.

II. Elements in Lightweight Cryptography

In this section, we briefly introduce the usual operations found in lightweight cryptographic primitives and protocols. This will serve to motivate our subsequent footprint analysis for individual building blocks.

A. Basic Operations

As in the case of regular security functions, lightweight cryptographic primitives and protocols aim at providing basic constructions to guarantee properties such as the confidentiality, integrity and authenticity of data exchanged in communications. In this case, however, the shortage of resources in the platforms severely limits the sort of processing that can be afforded. Thus, most proposals attempt to rely only on a few bitwise operations (e.g., XOR, OR, AND, shifts and rotations) and inexpensive arithmetics such as addition modulo 2^m.

In [24] a sort of mapping called triangular functions (T-functions) was introduced. In particular, a T-function is a $p \times p$ mapping that does not propagate information. Thus, for each $0 \leq i \leq p$, the i-th bit of the output is a function of the current and previous input bits $(0, 1, \ldots, i)$ only. Bitwise operations, such as for example NOT, XOR, OR and AND, and many others found in modern processors (e.g. addition, subtraction and multiplications) are all T-functions. Furthermore, the composition of T-functions is also a T-function.

Secure cryptographic primitives and protocols cannot be designed by using T-functions only. A T-function has poor diffusion, as it does not propagate information from right to left and, besides, its period is predictable [25]. As a consequence, T-functions are not the only operations that are usually found on lightweight cryptographic algorithms. For instance, in [26], [27] sound arguments are given for mixing triangular and non-triangular functions in order to design more secure ultra-lightweight protocols.

One of the most common non-triangular function used in cryptography is the rotation operation. Rotations can be performed in several ways. For instance, $rot^*(x, y)$ applies a circular left shift to the bits of x by a number of positions given by $wht(y)$ (the Hamming weight of the second operand y). Alternatively, using the classical definition, $rot(x, y)$ applies a circular left shift to the bits of x by $y \bmod N$, where N represents the bit length of variables x and y. Choosing a particular N determines the lightweight nature of this operation. For example, if N is a multiple of 2^n, then $rot(x, y)$ can be implemented very efficiently since it reduces to shifting the first argument n positions to the left; otherwise, it becomes more complex and requires a larger footprint area.

B. Storage and Control

Apart from arithmetic and logical operations, cryptographic algorithms also require additional hardware resources to store results and control the execution flow. Such elements are nearly always registers and multiplexers. Registers help to maintain the "state" of the algorithm by storing intermediate and final results, and also by supporting the control functions. Multiplexers are used to select among different inputs according to some condition, and are instrumental in any algorithm that incorporates a minimum flow complexity (i.e., "for" and "while' loops, "if" conditions, etc.).

In general, the area occupied by registers and multiplexers is critical, and any good design should find a balance between the amount of basic operations and memory/control logic. It is quite common to find proposals where authors only analyze the complexity of their designs by just counting the amount of operations. While this might be useful to determine the time complexity of the algorithm, ignoring memory and control requirements could be very misleading in terms of the final footprint area of the circuit. In fact, in many cases the area required by these elements is much larger than that demanded by the arithmetic and logical components.

III. Hardware Implementation of Basic Operations

We next analyze various design elements and their area estimation depending on the complexity of their implementation. For low-complexity blocks, we propose a simple architecture, while for those with higher complexity we discuss and propose several possible architectures.

A. Elements with a Low Complexity Implementation

In this categroy we include elements that have a straightforward implementation with very low complexity. We consider here some T-functions, such as simple bitwise operations and addition $mod\ 2^m$, and also registers and multiplexers.

In terms of bit length, it is common to find implementations tailored for 96 bits, as it is a common bit length widely used in low-cost RFID tags such as for example those compliant with EPC Gen-2 standard. However, we choose to study these elements specifically for arbitrary bit lengths, expressing the results as a function of a constant K. These constants will subsequently be used to obtain area estimations for more complex constructions.

For the above mentioned bitwise operations, we explore different design strategies that trade area off against throughput. In general, such strategies consist of using a block with a reduced bit length and consuming several clock cycles to obtain the final result. For example, an algorithm using an N-bit XOR block can be reduced to an $n/2$-bit block that needs two clock cycles to obtain the result. We note, however, that such block reductions involve the use of additional multiplexers for the control logic. Thus, it is necessary to find a trade-off between the reduction and the necessary extra logic. Moreover, reductions incur a drop in throughput, which should be properly accounted for if we are to met any restrictions imposed by the operational environment. For instance, taking as reference the performance criteria of an RFID system that demands a minimum reading speed of at least 150 tags per second [28], [29], we need to carefully calculate the affordable block reduction to fulfil this reading rate requirement. We

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2013.2288576, IEEE Transactions on Industrial Informatics

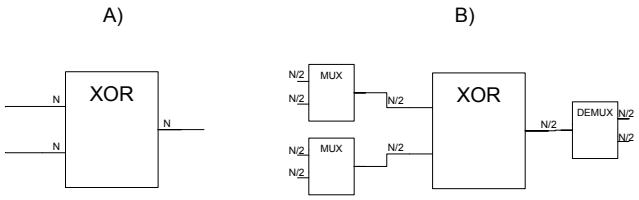

Fig. 1. Architectures for an XOR block.

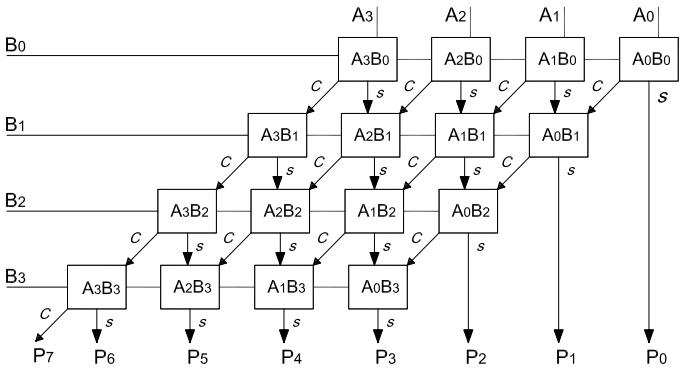

Fig. 2. Combinational Multiplier Architecture.

finally note that this strategy can be also used with other operations such as, for example, additions.

For illustration purpuses, in Fig. 1(a) we show the scheme of an N-bit XOR block. Fig. 1(b) shows the result after halving the XOR block ($N/2$ bits) and introducing additional multiplexers to select inputs and outputs.

B. Multiplication Operation

In this section we explore different hardware architecture to implement multiplication, including 1) a combinational architecture; 2) a shift-and-add architecture; and 3) the Karatsuba-Ofman architecture [30], [31].

1) Classical Combinational Multiplication: The most straightforward approach to implement a multiplier requires combinational logic only. The overall multiplication can be split into various partial multiplications followed by additions to sum the partial results. This alternative is the fastest in terms of throughput, but it has a high cost regarding the size of the circuit. Fig. 2 shows the needed hardware to implement a 4×4 bit multiplier of unsigned binary operands.

In Fig. 3, we show the internal structure of a basic cell. This cell includes and AND gate that computes the product of each bit of the multiplier q_j with the corresponding bit of the multiplicand m_j. The output of this product is one of the inputs to a Full Adder (FA), the other two operands being the corresponding bit from the previous partial product (Pp_i) and the carry (c) generated in the previous stage.

2) Classical Shift and Add Multiplication: The hardware footprint required by the combinatorial approach can be reduced by iterating the multiplication of the multiplicand by

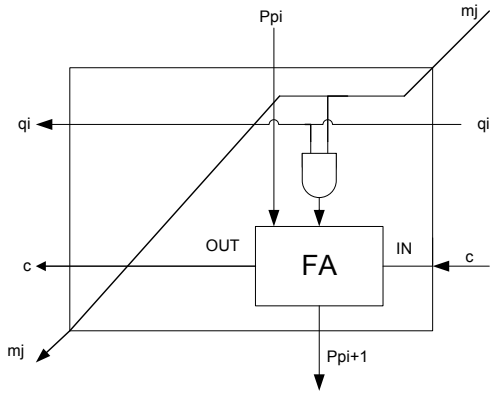

Fig. 3. Basic Cell of Combinational Multiplier.

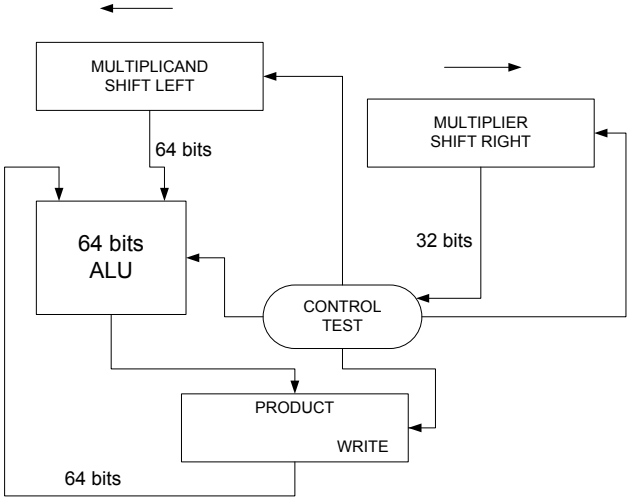

Fig. 4. Shift and Add Multiplier.

each bit of the multiplier and storing the partial results in a register. Since in each iteration the multiplier is a power of two, the partial multiplications can be implemented by just left shifting a register, which is very efficient in terms of clock cycles. To complete the architecture, as shown in Fig. 4, we need an adder to sum all the partial multiplications and a control unit to supervise the whole process. This approach turns out to be very efficient in terms of consumed hardware, but offers a low throughput due to its iterative nature. More precisely, assuming N-bit operands, this architecture consumes N clock cycles.

3) Karatsuba-Ofman: The Karatsuba-Ofman algorithm [30] achieves a trade-off between the chip area and the consumed clock cycles by dividing the $2n$-bit multiplication problem into smaller n-bit multiplication sub-problems, plus several n-bit additions and subtractions. Assume two $2n$-bit integers X and Y. Each integer can be split into two n-bit parts: a highest part (i.e., X_H and Y_H) and a lowest part (i.e., X_L and Y_L). Now, X and Y can be expressed in terms of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2013.2288576, IEEE Transactions on Industrial Informatics

their respective parts as:

$$X = X_H \cdot 2^n + X_L \quad (1)$$
$$Y = Y_H \cdot 2^n + Y_L \quad (2)$$

The product of X and Y can be now rewritten in terms of X_H, Y_H, X_L, and Y_L as:

$$X \cdot Y = (X_H \cdot 2^n + X_L) \cdot (Y_H \cdot 2^n + Y_L) \quad (3)$$
$$= 2^{2n}(X_H Y_H) + 2^n(X_H Y_L + X_L Y_H) + X_L Y_L$$

Note that the second term in expression (3) can be, in turn, represented in terms of the first and third terms:

$$X_H Y_L + X_L Y_H = \quad (4)$$
$$(X_H + X_L)(Y_H + Y_L) - X_H Y_H - X_L Y_L$$

In summary, the Karatsuba-Ofman algorithm computes the multiplication of two $2n$-bit operands by calculating three n-bit multiplications as:

$$X \cdot Y = 2^{2n}(X_H Y_H) + X_L Y_L + \quad (5)$$
$$2^n(X_H + X_L)(Y_H + Y_L) - X_H Y_H - X_L Y_L$$

We have analyzed two different architectures for implementing the Karatsuba-Ofman multiplication, which mainly differ on the mechanism used to compute each n-bit multiplications. The first one relies on an n-bit combinational multiplication and calculates each multiplication in one clock cycle. The second architecture uses an n-bit shift and add multiplication, which is more efficient in terms of chip area but takes several clock cycles to complete each multiplication.

C. Modulo Reduction

Modulo reductions are common in modular multiplications, and also appear in rotations like the $rot^*(x, y)$ discussed above. The hardware needed for its implementation depends on the value of the modulo. Given two positive numbers P (dividend) and N (divisor), $P \ mod \ N$ outputs the remainder of dividing P by N. This operation is very lightweight when N is a multiple of 2^n, since in that case the division reduces to various right shifts only. When this is not the case, a more general algorithm is required.

The modulo operation involves computing a division, which is an operation more complex than the multiplication. One straightforward algorithm is the so-called Naive Reduction, which shifts and subtracts the modulus until the remainder is obtained. Implementing Naive Reduction requires a subtractor, a comparator and an n-bit register, but it consumes a large amount of clock cycles (2^n). A different alternative is offered by a procedure known as Non Restoring Reduction (see Fig. 5), which is much more efficient in terms of clock cycles (approximately n instead of 2^n) but consumes more hardware.

Some algorithms apply a modulo reduction to the result of a multiplication. There are special implementations to optimize these combined operations, often involving the Montgomery modular multiplication algorithm with some convenient architecture [32]. In any case, these operations cannot be regarded as "lightweight" as we understand the term in this paper, so we will not study them in detail.

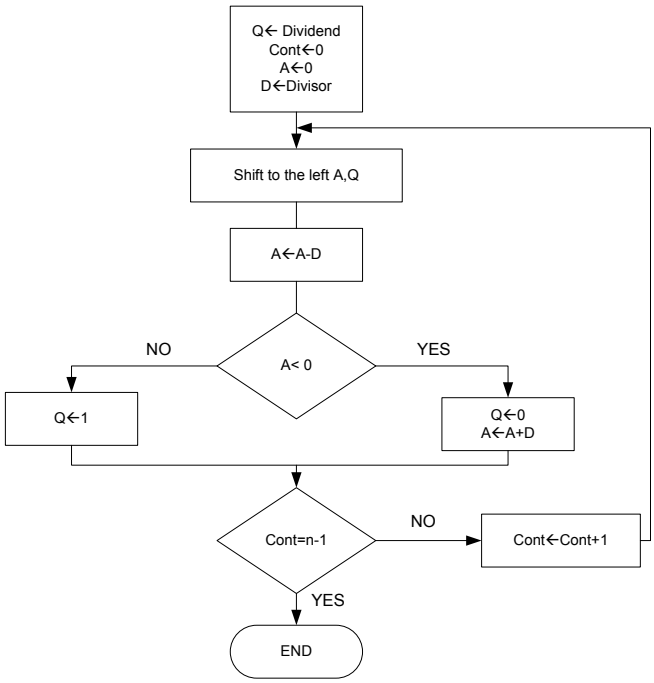

Fig. 5. Non Restoring Reduction Algorithm.

IV. Area Results for Low-complexity Elements

As discussed above, one major goal of this work is to provide an estimation of the area required by a lightweight cryptographic algorithm as a function of some high-level parameters. In these applications, it is crucial to keep in mind that circuits predominantly operate at low frequencies. For instance, many RFID tags function at 100 KHz. Note that the 100 kHz frequency refers to the clock included in the tag circuit, not to the communication band that is generally in the 860-960 MHz range for EPC C1-G2 tags. As the clock frequency is fixed, most restrictions in these designs relate to area and power consumption.

In this paper, we report results obtained with two specific manufacturing libraries. A priori, it is unclear to us the extent to which our conclusions generalize to other manufacturing technologies. The results, however, are still useful to compare different algorithms and classify them as lightweight or not. Furthermore, the methodology is general and can be easily extended to other libraries.

A. Experimental Framework

The experimentation has been conducted with two CMOS libraries: Faraday UMC 90 nm [33] and AMI 0.35 μm [34]. One key reason behind this decision is that these libraries provide comprehensive information about the layout of basic cells. For our purposes, this is essential to obtain a realistic estimate of the area occupied by an algorithm. To synthesize each design we used Synopsys [35], which is one of the most commonly used synthesizers.

The operation frequency is set to 100 KHz. As mentioned before, this is quite a common value for passive RFID tags.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2013.2288576, IEEE Transactions on Industrial Informatics

TABLE I
GEs FOR LOW-COMPLEXITY ELEMENTS

Lib.	Element	32 bits	64 bits	96 bits	128 bits	$\mathbf{K_i}$
UMC 90nm	AND	39.70	79.39	119.08	158.78	K_1=1.24
	OR	39.70	79.39	119.08	158.78	K_2=1.24
	XOR	79.39	158.78	238.17	317.56	K_3=2.48
	ADD	239.66	477.84	716.01	954.19	K_4=7.45
	Multiplexer	71.00	143.00	214.00	285.81	K_{mux}=2.23
	Register	147.00	287.00	441.00	588.24	K_{reg}=4.59
AMI 0.35 μm	AND	42.66	85.33	127.99	170.64	K_1=1.33
	OR	53.33	106.65	160.03	213.22	K_2=1.66
	XOR	74.66	149.34	224.00	298.70	K_3=2.33
	ADD	203.35	406.00	608.69	811.32	K_4=6.34
	Multiplexer	85.33	170.66	256.00	341.55	K_{mux}=2.66
	Register	214.33	435.66	651.66	869.00	K_{reg}=6.77

Power supply is fixed by the library (1.2 V). As for the synthesis with Synopsys, after experimenting with different configurations we observed that the best results are obtained with the *medium effort* option in area, delay and power consumption. These options are set for all the experiments.

Finally, the area results are provided using Gate Equivalents (GEs), which is the normalization commonly used for these applications. Using GEs facilitates comparisons among different implementations since the obtained values are independent of the chosen technology. To compute the GE value, the area of the whole circuit is divided by the area of a basic NAND gate. For example, 1 GE for the UMC 90 nm takes 3.16 μm^2.

B. Results

Table I summarizes the area results (in GEs) obtained after synthesizing with Synopsis the set of basic elements for UMC 90 nm and AMI 0.35 μm libraries. In this first analysis, the hardware architecture considered performs all operations (combinational or just registers) in one clock cycle. As shown in Fig. 6, the area occupied by each element increases linearly in the length (in bits) of variables. The results obtained for the AMI 0.35 μm library are almost equivalent and follow a similar pattern. This simplifies considerably the analysis of more complex algorithms, as it allows us to associate a constant value, named K_i for element i, giving the area per bit for each element.

From these results we can extract some conclusions:

1) The adder occupies significantly more area than bitwise operations. Consequently, if the area of an algorithm needs to be optimized, it is more appropriate to focus on additions rather than concentrating on low complexity elements such as bitwise operations. As all operations are done in one clock cycle, one possibility to optimize the area would be to use an element with lower bit length and carry out the operation in various clock cycles. For example, variables can be split into two parts with half of the bits each and a half-length adder can then be applied over each part. Note, however, that in doing this we need to include additional elements, namely registers to store partial results, multiplexers to choose between different signals, etc.
2) The area cost of registers is also noticeable. Taking into account that we generally can devote just a small area to security subsystems (e.g., up to 4K GEs in most passive RFID tags), and that roughly 50% of it is used for storage, this means that at most five 96-bit registers could be used.
3) As for multiplexers, their cost in terms of area is small. These elements are needed in algorithms with loops (e.g., "for" and "while" iterations) and also when a input/output is selected among different signals.

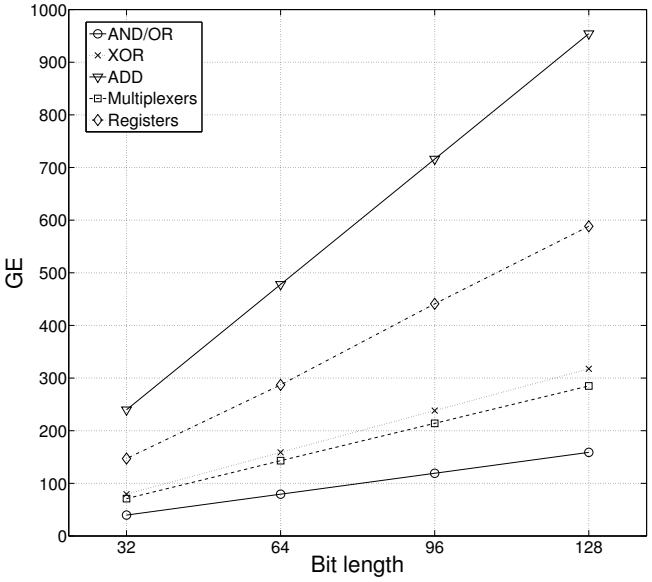

Fig. 6. GE for low complexity elements as a function of the number of bits for UMC 90nm library.

Overall, it can be concluded that designers will necessarily face some trade-offs among operations and the amount of registers and multiplexers required. As a general rule, bigger building blocks (i.e., using a larger bit length) will require less extra registers/multiplexers, and vice versa.

C. Further Operations: Multiplication and Modulo Reduction

We next explore the area required by two operations that have been extensively used in many cryptographic algorithms: multiplication and reduction modulo N. The figures, both the number of GEs and the associated clock cycles required to complete the operation, are shown in Table II for the UMC 90 nm and the AMI 0.35 μm libraries.

In general terms, multiplication cannot be regarded as a lightweight operation no matter what architecture is chosen, since it demands more than the 4K GEs often required in environments such as RFID systems (96 bits). That being said, it is worth-noting that some trade-offs also appear here. The combinational architecture offers the best performance speed-wise, but it demands too much area. Conversely, the Shift-and-Add option is much more efficient in terms of area, but the number of clock cycles requires may be prohibitive for many applications. K-O architectures fall somewhere in between of these two alternatives.

Modulo reduction is a special case. As discussed before, it is very lightweight when the bit length N is a power of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2013.2288576, IEEE Transactions on Industrial Informatics

TABLE II
GEs FOR DIFFERENT MULTIPLICATION ARCHITECTURES AND MODULO REDUCTION

Lib.	Operation	32 bits	64 bits	96 bits	128 bits	Cycles
UMC 90nm	MULT (Comb.)	9345	36507	81268	144452	1
	MULT (S+A*)	2078	4113	6146	8164	N
	MULT (K-O†, Comb)	5744	16055	30853	49868	10
	MULT (K-O†, S+A*)	4731	9367	13995	18566	$\frac{N}{2}+4$
	Modulo reduction	–	–	3967	–	96
AMI 0.35 μm	MULT (Comb.)	10223	36495	81007	143436	1
	MULT (S+A*)	2464	4840	7227	9639	N
	MULT (K-O†, Comb)	6621	17093	32123	51132	10
	MULT (K-O†, S+A*)	5882	11610	17341	23111	$\frac{N}{2}+4$
	Modulo reduction	–	–	4729	–	96

†K-O: Karatsuba-Ofman multiplier *S+A: Shift-Add multiplier

two, as it can be implemented simply as various right shifts. Otherwise, such as for example for $N = 96$, its area takes around 4K GEs. Thus, our recommendation is to include it only when the resources required by this operation can be reused in other parts of the algorithm.

V. ESTIMATING THE AREA OF LIGHTWEIGHT ALGORITHMS

Estimating the area that an implementation of an algorithm can occupy is quite challenging because it depends on many factors: the architecture(s) chosen by the designer, the specific constraints, the manufacturing library, the basic cells used by the synthesis tool, etc. In this section, we first propose an expression that estimates the total area required by a hardware implementation of an algorithm. Subsequently, we check its validity by comparing its predictions with the actual area obtained with a battery of examples and provide a refinement of our estimator. Note that we have discarded the use of multiplication since this operation consumes resources in excess (>4K GEs) to be categorized as a lightweight operation. Regarding modulo reduction, its usage in a lightweight algorithm is conditioned to be a power of two (i.e. 2^n); otherwise it demands more than 4K GEs and using it is infeasible. In case of being a power of two, the operation does not consume any extra hardware resources, but requires an upper bound of n clock cycles to compute it.

A. A Linear Estimator

The total area occupied by an algorithm can be roughly divided into two main blocks: datapath and control. The datapath contains the hardware for the different operations required and registers to store inputs, outputs and intermediate results. In many lightweight cryptographic algorithms, the datapath accounts for a significant fraction of the total area, generally around 80% [36] [37].

Our estimate is based on the following rationale. As we previously pointed out, the final footprint depends on the chosen architecture. In turn, opting for one architecture or another depends on the goals and restrictions faced by the designer. For example, in very constrained devices (such as RFID tags or some sensor nodes) minimizing the area is a priority, which heavily influences the decision. Since throughput is often a limiting factor too, one sensible choice is an architecture that optimizes the area without penalizing throughput too much. In general, such a design contains one single block of N bits for each basic operation needed, plus registers to store data and multiplexers to select inputs and outputs.

Based on the previous considerations, we propose a simple linear estimate for the area of the datapath, measured in GE, as a function of the bit length and the number of basic operations, registers and multiplexers:

$$F_{DP} = N \cdot \left[\sum_{i=1}^{4} A_i \cdot K_i + (B \cdot K_{reg}) + \left((C+D) \cdot K_{mux} \right) \right] \quad (6)$$

where:

- N is the bit length of the variables.
- A_i is a parameter dependent on the chosen architecture for the datapath ($i = 1$ for AND, $i = 2$ for OR, $i = 3$ for XOR, and $i = 4$ for ADD). As discussed above, the implementation can range from a fully combinational design to one using smaller operators but requiring more clock cycles. Thus, we measure A_i as the number of N-bit operators.
- K_i is the area cost for the i-th operation, as shown in Table I.
- B is the number of variables that require storage.
- K_{reg} is the area cost for each register.
- C is the number of multiplexers necessary to select different inputs for the operation blocks. When the block has more than two inputs, C is the number of inputs minus one.
- D is the number of multiplexers necessary to select different inputs for each register. If the algorithm is given in pseudocode, D can be easily estimated as the number of assignments made for each variable.
- K_{mux} is the area cost for the multiplexers.

Obviously, expression (6) only factors in those elements studied in Section IV. However, it can be extended without difficulty to any other blocks that conform to the design rationale given in the paragraph above.

Finally, as the area of datapath and control are in most cases related, we express the total area as:

$$F = (1 + \omega) \cdot F_{DP} \quad (7)$$

where ω is an *overhead* factor accounting for the control part (e.g., $\omega = 0.2$ assuming that control logic accounts for 20% of the total area).

B. Experimental results

We have tested our estimator against a library containing 120 lightweight functions. The algorithms are named $F_1, F_2, \ldots, F_{30}$ and were synthesized for four different bit lengths: $N = 32$, 64, 96, and 128 bits. Each function returns a single final output value denoted Z and uses several input and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2013.2288576, IEEE Transactions on Industrial Informatics

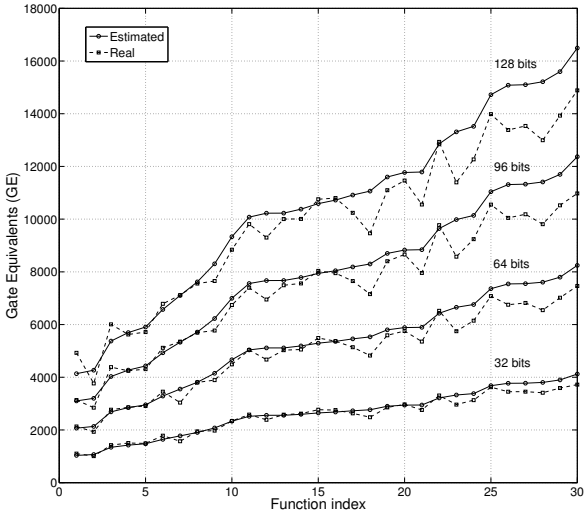

Fig. 7. Real and estimated footprint area ($\omega = 0.2$).

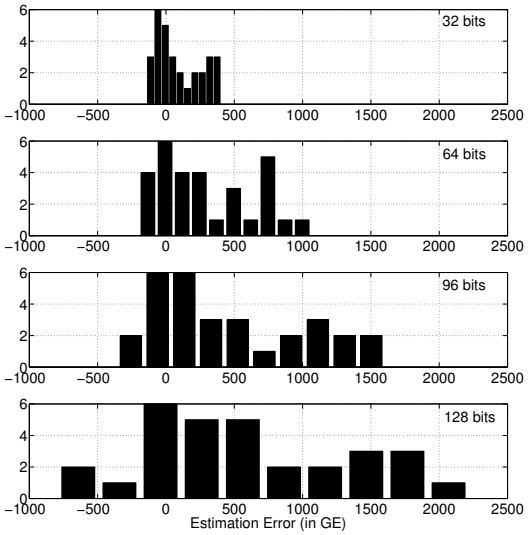

Fig. 8. Distribution of gate count estimation errors ($\omega = 0.2$).

intermediate variables, represented by X_i and Y_i, respectively. The dataset is well balanced, containing 10 functions with 2 inputs, 10 functions with 4 inputs, and another 10 functions with 6 inputs.

In Fig. 7 we compare the estimated area for all the datasets functions using (7), assuming a control overhead $\omega = 0.2$, versus the actual area given by Synapsis after synthesizing each function. For simplicity, we only show the results obtained for the UMC 90 nm library. As suggested by Tables I and II, the results for the AMI 0.35 μm are completely equivalent, and our experimentation confirms this. The approximation is quite precise, with differences becoming greater when the number of bits N increases. In Fig. 8 we show histograms of the estimation errors for different bit lengths. For $N = 32$ and 16 bits, errors are bounded by 500 GE and 1K GEs respectively. This error increases to 1.5K GE and 2K GEs for $N = 96$ and 128 bits, respectively. Thus, chosing a high value for the control overhead (20%) in Equation 7 does not minimize errors but guarantees an overestimation of circuit area.

Further investigations reveal that the overestimation does not come from expression (6), but from (7). In other words, the estimate for the datapath area is fairly accurate, but the amount of control logic does not generally increase linearly with the number of bits. For instance, a Finite State Machine (FSM) controlling some parts of an algorithm does not need more states when variables increase their size. That being said, we emphasize that our choosing of (7) may still be valid for constrained designs, where N often varies between 32 and 512, interpreting the result as an upper bound.

C. Adjusting control overheads

As discussed in Section V-A, the datapath and control areas are in most cases related. In the model presented above we made the assumption that the relation is linear, in particular with the control logic being a fraction $(1+\omega)$ of the datapath area. The experimental results discussed above show that this assumption works relatively well for systems of up to 10K GE, particularly with $\omega = 0.2$. The estimation error becomes more significant for bigger systems. This is reasonable, as as increase in the datapath footprint does not necessarily translate into a similar increase of control logic.

Using the dataset of designs described above, we have numerically investigated more precise approximations for the control overhead term used in (7). Two alternatives were explored, both based on the idea that ω varies with some system parameter. In the first one, we assumed that the control overhead depends on the number of bits N, so the total area is actually of the form:

$$F = \left[1 + \omega(N)\right] \cdot F_{DP} \tag{8}$$

whereas in the second alternative it is assumed that the amount of control logic is a function of the datapath area:

$$F = \left[1 + \omega(F_{DP})\right] \cdot F_{DP} \tag{9}$$

The estimation of both functions $\omega(N)$ and $\omega(F_{DP})$ was done by couching the problem as a nonnegative least-squares curve fitting one of the form:

$$\min_{\Omega} \|C\Omega - d\|_2^2 \tag{10}$$

where $\Omega = (\omega_1, \ldots, \omega_k)^T$, with $\omega_i \geq 0$, represents the sought function discretized in k values. Matrix C and vector d contain, respectively, the *actual* datapath area and total area obtained after synthesis.

We split the dataset of designs into two subsets. The first one, used to estimate the overhead function (training) contains 80 randomly chosen (10 of each bit length) designs out the 120 available. The remaining 40 designs will be subsequently used to test the obtained estimator. Thus, each one of the 80 synthesized functions used for training gives one equation for

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2013.2288576, IEEE Transactions on Industrial Informatics

TABLE III
NUMERICALLY ESTIMATED CONTROL OVERHEAD FUNCTIONS

N	$\omega(N)$
32 bits	0.1518
64 bits	0.1186
96 bits	0.1192
128 bits	0.1103
$\|\text{residual}\|_2^2$	3.36E+06

F_{DP}	$\omega(F_{DP})$
0 - 2000 GE	0.1906
2000 - 4000 GE	0.1717
4000 - 6000 GE	0.1691
6000 - 8000 GE	0.1271
8000 - 10000 GE	0.1098
10000 - 12000 GE	0.0932
12000 - 14000 GE	0.0774
$\|\text{residual}\|_2^2$	1.38E+05

(10), which are grouped into k bins. In the case of $\omega(N)$, we chose $k = 4$ values (32, 64, 96, and 128 bits), whereas for $\omega(F_{DP})$ we grouped equations into $k = 7$ intervals with a 2K GE difference between each of them.

Using a standard numerical solver, we obtained the Ω-values shown in Table III. Again, these figures correspond to the UMC 90 nm library; those obtained for the AMI 0.35 μm are very similar. Such overheads represent the best fit, in a least-squares sense, for our experimental dataset. As observed, in both cases the actual overhead is always below the fixed $\omega = 0.2$ value that was used before. Furthermore, it decreases as circuits grow bigger, both in terms of N and in datapath area, which conforms to our previous intuition. For example, in systems with less than 4K GE the overhead accounts for 16%-19% of the datapath area, but it falls down to less than 10% when the datapath is 10K GE or more. This is also observed when the overhead is considered a function of N.

Analysis of the squared 2-norm of the residual reveals that the $\omega(F_{DP})$ estimation performs significantly better than $\omega(N)$. Thus, while the former yields a squared residual of 1.38E+05, which roughly translates into an average error of 371 GE per design, the latter is greater by more than an order of magnitude (3.36E+06), meaning an error of around 1833 GE per estimation. This is also reasonable, as it appears to be more sensible that the amount of control logic depends more on the datapath area rather than on the length of registers.

Overall, using functional overheads such as these provide us with a more precise estimation of the total footprint area. For comparison with the plots discussed in previous section, Figs. 9 and 10 show the adjusted estimates for the training and test functions, respectively. Similarly, Fig. 11 shows the error distribution over test functions only. It is clear that the fit is now much more accurate (compare with Fig. 8), even though the new estimation cannot be regarded anymore as an upper bound for the total footprint area.

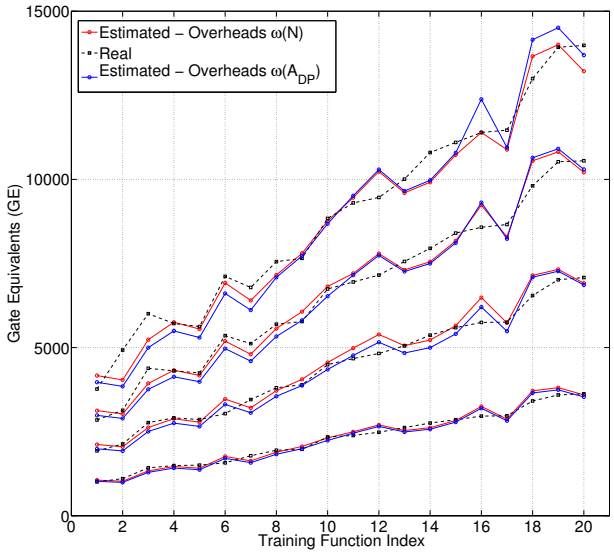

Fig. 9. Real and estimated footprint area using adjusted control overheads: results on 80 training designs.

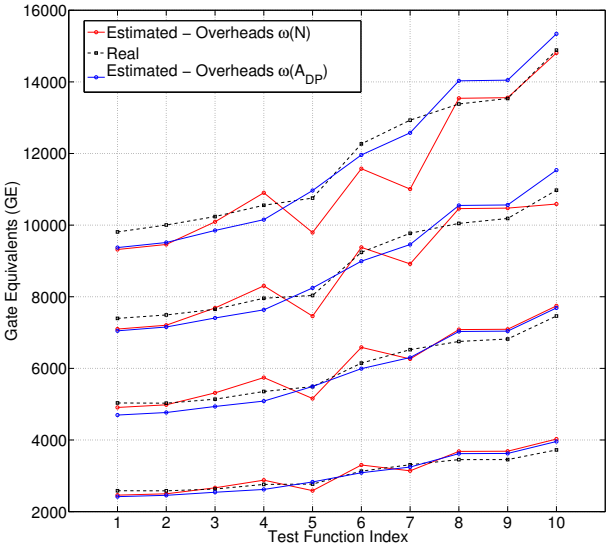

Fig. 10. Real and estimated footprint area using adjusted control overheads: results on 40 test designs.

VI. CONCLUSIONS

In this paper, we have proposed a simple yet accurate procedure to estimate the footprint area of generic lightweight algorithms. We have argued that finding an accurate approximation is extremely hard, since it strongly depends on factors such as the architecture chosen by the designer, the manufacturing technology, the libraries used, the possibility of optimizing the footprint when combining several parts, etc. Despite this, the designer of algorithms for constrained environments (such as, for example, those related to cryptographic functions for RFID tags or sensor nodes) should count on some quantities to drive their choices. One major motivation for this work is to fill this gap by providing algorithm designers with a tool to estimate the cost, in terms of footprint area, of their constructions.

We believe our proposal will help in making some choices at the algorithmic level, even for designers without hardware design skills. Furthermore, it could also be applied to get preliminary comparisons among different proposals (lightweight primitives and more complex constructions such as security protocols) or, at least, to decide if they are simply too costly for certain operational environments.

The work presented in this paper can be extended in a number of ways. One natural direction for future work is the inclusion of other commonly used elements in the F_{DP}

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2013.2288576, IEEE Transactions on Industrial Informatics

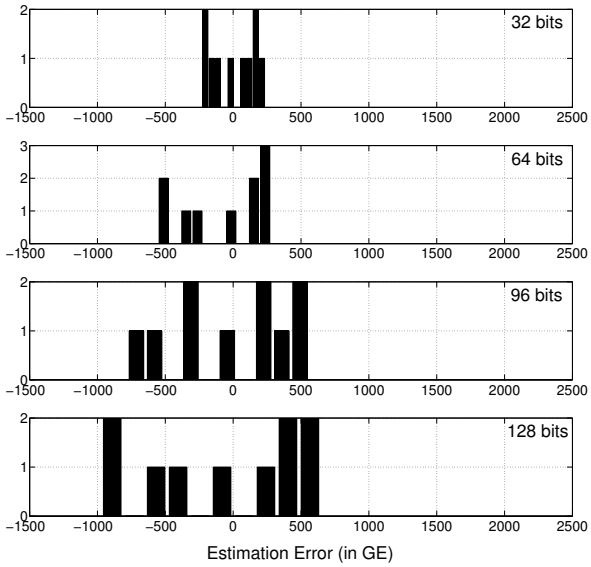

Fig. 11. Distribution of gate count estimation errors using $\omega(F_{DP})$ on 40 test designs.

estimator, such as for example S-boxes of non-linear filters. Similarly, we expect to test the proposed estimator against well-known lightweight cryptographic primitives and compare the predictions with reported experimental results. Finally, our focus in this work has been exclusively on the footprint area of ASIC implementations. It would be interesting to extend our estimates to include other prominent parameters, primarily throughput and power consumption, as these have also significant influence in design choices. Consider, for example, one the most used RFID standards [4], [5], where a tag must support up to 1500 read attempts per second under ideal conditions, although this rate can be five or ten times smaller (500-150 tags/sec) in real world environments [29]. Therefore, for a tag operating frequency of 100 KHz, the number of clock cycles consumed per reading is upper-bounded by 670 (in fact, 500 clock cyles is an upper bound commonly assumed in previous works [37], [38]). In principle, the methodology discussed in this work can be easily extended to incorporate measures of throughput and power consumption. We expect to tackle this in future work.

REFERENCES

[1] S. Park and Hongchul Lee. "Self-Recognition of Vehicle Position Using UHF Passive RFID Tags," *IEEE Trans. on Industrial Electronics*, vol. 60, no. 1, pp. 226-234, 2013.

[2] D. Maimut and K. Ouafi. "Lightweight Cryptography for RFID Tags," *IEE Security & Privacy*, vol. 10, no. 2, pp. 76-79, 2012.

[3] S. Chia, A. Zalzala, L. Zalzala, A. Karim. "Intelligent Technologies for Self-Sustaining, RFID-Based, Rural e-Health Systems," *IEEE Technology and Society Magazine*, vol.32, no. 1, pp. 36-43, 2013.

[4] EPCglobal. "EPC Radio-Frequency Identity Protocols. Class-1 Generation-2 UHF RFID. Protocol for communications at 860 MHz - 960 MHz," (Version 1.2.0), 2008.

[5] ISO/IEC 18000-6. "Information technology – Radio Frequency Identification for item management – Part 6: Parameters for air interface communications at 860 MHz to 960 MHz," 2013.

[6] B. Wang and M. Ma. "A Server Independent Authentication Scheme for RFID Systems," *IEEE Trans. on Industrial Informatics*, vol. 8, no. 3, pp. 689-696, 2012.

[7] A. Abu-Mahfouz and G.P. Hancke. "Distance Bounding: A Practical Security Solution for Real-Time Location Systems," *IEEE Trans. on Industrial Informatics*, vol. 9, no. 1, pp. 16-27, 2013.

[8] T. Plos, M. Hutter, M. Feldhofer, M. Stiglic, and F. Cavaliere. "Security-Enabled Near-Field Communication Tag With Flexible Architecture Supporting Asymmetric Cryptography," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, In Press.

[9] B. Fabian, T. Ermakova, and C. Muller. "SHARDIS: A Privacy-Enhanced Discovery Service for RFID-Based Product Information," *IEEE Trans. on Industrial Informatics*, vol. 8, no. 3, pp. 707-718, 2012.

[10] S. Piramuthu. "RFID mutual authentication protocols," *Decision Support Systems*, vol. 50, no. 2, pp. 387-393, 2011.

[11] P. Yang, W. Wu, M. Moniri, and C. C. Chibelushi. "Efficient Object Localization Using Sparsely Distributed Passive RFID Tags," *IEEE Trans. on Industrial Electronics*, vol. 60, no. 12, pp. 5914-5924, 2013.

[12] C. C. Tan and Q. L. Bo Sheng. "Secure and Serverless RFID Authentication and Search Protocols," *IEEE Trans. on Wireless Communications*, vol. 7, no. 4, pp. 1400-1407, 2008.

[13] A. Vena, E. Perret, and S. Tedjini. "Design of compact and auto-compensated single-layer chipless RFID tag," *IEEE Trans. on Microwave Theory and Techniques*, vol. 60, no.9, pp. 2913-2924, 2012.

[14] J. Li and S. M. R. Hasan. "Design and Performance Analysis of a 866-MHz Low-Power Optimized CMOS LNA for UHF RFID," *IEEE Trans. on Industrial Electronics*, vol. 60, no. 5, pp. 1840-1849, 2013.

[15] C. Wang, M. Daneshmand, K. Sohraby, and B. Li. "Performance analysis of RFID Generation-2 protocol," *IEEE Trans. on Wireless Communications*, vol. 8, no. 5, pp. 2592-2601, 2009.

[16] S. G. Baskir and B. Ors. "Implementation of a secure RFID protocol," in *Signal Processing and Communications Applications Conference (SIU)*, pp. 1-4, 2013.

[17] S. Hosseini-Khayat. "A lightweight security protocol for ultra-low power ASIC implementation for wireless Implantable Medical Devices," in *5th International Symposium on Medical Information & Communication Technology (ISMICT)*, pp. 6-9, 2011.

[18] X. Guo, M. Srivastav, S. Huang, D. Ganta, M. B. Henry, L. Nazhandali, and P. Schaumont. "ASIC implementations of five SHA-3 finalists," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1006-1011, 2012.

[19] A. Kempitiya, D.-A. Borca-Tasciuc, and M. M. Hella. "Low-Power ASIC for Microwatt Electrostatic Energy Harvesters," *IEEE Trans. on Industrial Electronics*, vol. 60, no. 12, pp. 5639-5647, 2013.

[20] S.-Y. Lee, L.-H. Wang and Q. Fang. "A Low-Power RFID Integrated Circuits for Intelligent Healthcare Systems," *IEEE Trans. on Information Technology in Biomedicine*, vol. 14, no. 6, pp. 1387-1396, 2010.

[21] Y. Zhou and C. L. Law and J. Xia. "Ultra low-power UWB-RFID system for precise location-aware applications," in *IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pp. 154-158, 2012.

[22] J. A. Rodriguez-Rodriguez, M. Delgado-Restituto, J. Masuch, A. Rodriguez-Perez, E. Alarcon, and A. Rodriguez-Vazquez. "An Ultralow-Power Mixed-Signal Back End for Passive Sensor UHF RFID Transponders," *IEEE Trans. on Industrial Electronics*, vol. 59, no. 2, pp. 1310-1322, 2012.

[23] R. Ben Atitallah, E. Senn, D. Chillet, M. Lanoe, and D. Blouin. "An Efficient Framework for Power-Aware Design of Heterogeneous MPSoC," *IEEE Trans. on Industrial Informatics*, vol. 9, no. 1, pp. 487-501, 2013.

[24] A. Klimov and A. Shamir. "A new class of invertible mappings". In *Cryptographic Hardware and Embedded Systems (CHES)*, vol. 2523 of LNCS, pp. 471-484, 2002.

[25] G. Avoine, X. Carpent, B. Martin. "Privacy-friendly synchronized ultra-lightweight authentication protocols in the storm", *Journal of Network and Computer Applications*, vol. 35, n. 2, pp. 826-843, 2012.

[26] H.-Y Chien. "SASI: A new ultralightweight RFID authentication protocol providing strong authentication and strong integrity," *IEEE Trans. on Dependable and Secure Computing*, vol. 4, no. 4, pp. 337-340, 2007.

[27] P. D'Arco and A.P. de Santis. "On Ultralightweight RFID Authentication Protocols," *IEEE Trans. on Dependable and Secure Computing*, vol. 8, no. 4, pp. 548-563, 2011.

[28] D, Ranasinghe and P. Cole. "An evaluation framework," *Networked RFID Systems and Lightweight Cryptography*, ch. 8, pp. 157-167, 2007.

[29] M. Brown, E. Zeisel, and R. Sabella, "Chapter 2 - RFID tags," in *RFID+ Exam Cram*. Que, 2006.

[30] N. Nedjah and L. de M. Mourelle. "A review of modular multiplication methods and respective hardware implementation," *Informatica (Slovenia)*, vol. 30, no. 1, pp. 111-129, 2006.
[31] M.-D. Shieh, J.-H. Chen, W.-C. Lin and H.-H. Wu. "A new algorithm for high-speed modular multiplication design," *IEEE Trans. on Circuits and Systems*, vol. 56, no. 9, pp. 2009-2019, 2009.
[32] M. Huang, K. Gaj, and T. El-Ghazawi. "New Hardware Architectures for Montgomery Modular Multiplication Algorithm," *IEEE Trans. on Computers*, vol. 60, no. 7, pp. 923-936, 2011.
[33] 90nm Generic Core Cell Library. Data Book. Rev.: 0.3, 2009.
[34] AMI Semiconductor C035U CMOS. Design Rules. Rev.: A., 2006.
[35] Synopsys Design Compiler User Guide. Version D-2010.03-SP2, 2010.
[36] P. Peris-Lopez, J. C. Hernandez-Castro, J. E. Tapiador, and A. Ribagorda. "LAMED A PRNG for EPC Class-1 Generation-2 RFID specification", *Computer Standards & Interfaces*, vol. 31, no. 1, pp. 88-97, 2009.
[37] J. Melia-Segui, J. Garcia-Alfaro, and J. Herrera-Joancomarti. "Analysis and Improvement of a Pseudorandom Number Generator for EPC Gen2 Tags," in *Financial Cryptography and Data Security*, LNCS series, vol. 6054, pp. 34-46, 2010.
[38] K. Mandal, X. Fan, and G. Gong. "A lightweight pseudorandom number generator for EPC C1 Gen2 tags," in *Radio Frequency Identification System Security*, ser. Cryptology and Information Security, vol. 8, pp. 7384, 2012.

Honorio Martin is a Ph.D student at the Electronics Technology Department, Universidad Carlos III de Madrid, Spain. He holds a Research Master's Degree in Advanced Electronics Systems. His current research interests include the study of lightweight cryptography hardware implementations, Radio Frequency Identification (RFID) systems and low-power designs.

Pedro Peris-Lopez is Visiting Lecturer at the Department of Computer Science, Universidad Carlos III de Madrid, Spain. He holds a M.Sc. in Telecommunications Engineering and Ph.D. in Computer Science. His research interests are in the field of protocols design, primitives design, lightweight cryptography, cryptanalysis etc. Nowadays, his research is focused on Radio Frequency Identification Systems (RFID) and Implantable Medical Devices (IMD). In these fields, he has published a great number of papers in specialized journals and conference proceedings. For additional information see: www.lightweightcryptography.com/.

Juan E. Tapiador is Associate Professor at the Department of Computer Science, Universidad Carlos III de Madrid, Spain. He holds an M.Sc. (2000) and a Ph.D. (2004) in Computer Science from the University of Granada. Before joining UC3M, between 2009 and 2011 he was Research Associate at the University of York, UK. His main research interests are in applied cryptography and computer and network security. For additional information see: www.seg.inf.uc3m.es/~jet.

Enrique San Millan is Associate Professor in the Electronics Technology Department, Universidad Carlos III de Madrid, Spain. He holds an M.Sc. in Mathematics from La Rioja University (Spain) and a Ph.D in Mathematics Engineering from Universidad Carlos III de Madrid (Spain). His main research interests include hardware design of digital circuits and systems for several fields (cryptography, biometry, fault tolerant systems, communications) and CAD tools for design automation and optimization of digital integrated circuits.