information

Article

Hardware Support for Security in the Internet of Things: From Lightweight Countermeasures to Accelerated Homomorphic Encryption

Régis Leveugle [1,2,*], Asma Mkhinini [1] and Paolo Maistri [1]

[1] Univ. Grenoble Alpes, CNRS, Grenoble INP^2, TIMA, 38000 Grenoble, France; Asma.Mkhinini@univ-grenoble-alpes.fr (A.M.); Paolo.Maistri@univ-grenoble-alpes.fr (P.M.)
[2] Institute of Engineering, Univ. Grenoble Alpes, 38000 Grenoble, France
* Correspondence: Regis.Leveugle@univ-grenoble-alpes.fr; Tel.: +33-(0)476-574-686

Received: 29 March 2018; Accepted: 20 April 2018; Published: 8 May 2018

Abstract: In the Internet of Things (IoT), many strong constraints have to be considered when designing the connected objects, including low cost and low power, thus limited resources. The confidentiality and integrity of sensitive data must however be ensured even when they have to be processed in the cloud. Security is therefore one of the design constraints but must be achieved without the usual level of resources. In this paper, we address two very different examples showing how embedded hardware/software co-design can help in improving security in the IoT context. The first example targets so-called "hardware attacks" and we show how some simple attacks can be made much more difficult at very low cost. This is demonstrated on a crypto-processor designed for Elliptic Curve Cryptography (ECC). A very lightweight countermeasure is implemented against Simple Power Analysis (SPA), taking advantage of the general processor usually available in the system. The second example shows how confidentiality in the cloud can be guaranteed by homomorphic encryption at a lower computational cost by taking advantage of a hardware accelerator. The proposed accelerator is very easy to implement and can easily be tuned to several encryption schemes and several trade-offs between hardware costs and computation speed.

Keywords: IoT security; integrated circuits; hardware accelerators; crypto-processors; ECC encryption; homomorphic encryption; lightweight; SPA; countermeasure

1. Introduction

IoT (Internet of Things) is today one of the major worldwide change in the information society, with impacts in every domain, including smart cities and buildings, health, transport, energy, industrial processes, and the everyday life of a large part of humanity [1]. IoT is the result of a quick evolution, starting with the first automation of some processes, making use of electronic devices, then enhanced with communication capabilities. This led to SCADA (Supervisory Control And Data Acquisition) and M2M (Machine to Machine) for e.g., industrial, financial, healthcare, transport or government services. IoT is now covering a much larger application area, including the global evolution of city- or nation-level services as well as in consumer everyday-life features. Diversity is therefore the keyword and every new device must be designed having in mind the specific constraints of the targeted users. There is thus no single answer to secure, or help to secure, all applications.

In this evolution, one basis did not change. New services were proposed thanks to new electronic (hardware/software) devices embedded in an increasing part of the objects used both in the industrial and consumer markets. Due to the constraints on cost, size and power consumption (also related to autonomy for devices that have to be self-sufficient for a long time), enablers are mostly integrated circuits, either specific or off-the-shelf, such as microcontroller units (MCUs). In these circuits, dedicated blocks (so-called "hardware accelerators") are most often used in order to achieve a good trade-off between performance and e.g., power, in the case of the most demanding functions. Simpler functions are in general implemented in software and executed on an embedded general-purpose processor. We will show why such electronic devices must be protected against security attacks, on the basis of real-life examples e.g., [2,3]. The protections must in many cases come at low cost, at least to thwart the easiest attacks. Proposing a new approach for such cheap protections is the goal of the first part of our contribution. We will demonstrate how hardware/software partitioning can be exploited.

In the global evolution briefly sketched above, another aspect noticeably changed in only a few years, with an increasing use of cloud computing, shared data and world-wide cooperation through shared outsourced databases, fed by many distributed sources (or sensors). However, preserving data confidentiality is of upmost importance in many domains such as genomic analysis, healthcare or other applications using very private data. In this context, homomorphic encryption [4] has become a very appealing solution, but with a cost remaining today that is too high to satisfy the practical requirements in most cases. There is therefore a need to make these approaches more practical, without resorting to very large computing systems. We will show that once again relying on a partitioning between software and hardware accelerators can be a good way to increase efficiency.

In this paper, we will therefore illustrate on the basis of these two different aspects that developing specific hardware-based primitives associated with embedded software can lead to efficient systems in terms of security and/or in terms of implementation costs, performance and energy. In our opinion, all these aspects should be considered simultaneously in order to seek efficient implementations.

More detailed comparisons with existing approaches will be presented in the sequel, limiting the references to some significant examples in the literature. To summarize more broadly the state-of-the-art, for the protections against attacks either algorithmic/software techniques are used as mentioned in [3] or specific hardware elements are added to the hardware accelerators, using for example hardware redundancy [5]. We propose a different approach, mixing hardware and software, and running some kind of protection software in parallel to critical hardware computations. For the second aspect related to homomorphic encryption, most implementations only use software, as in [6], and a few implementations are only focused on hardware, as in [7–9]. Our contribution is based on a balanced partitioning between hardware and software. The main point of our global contribution is to demonstrate that, for very different aspects related to security, optimizations can be achieved considering such hardware/software partitioning. In addition to security improvements, other characteristics important in the IoT context can be optimized as well, such as power consumption and the execution time on low-cost devices.

The paper is organized as follows. In Section 2, the context will be described in more detail. In Section 3, we will focus on threats related to hardware-based attacks. We will show that at least the easiest attacks can be made much more complex at very low cost. In Section 4, we will discuss how data confidentiality can be ensured more efficiently with great flexibility using hardware acceleration and high-level synthesis (HLS).

2. Concrete Motivation Examples and the Need for Hardware Protections

We explain in this section (a few) examples demonstrating that security cannot be neglected in IoT, including hardware-based security. Many reports such as [1] indicate that this aspect can be a show-stopper in the development of IoT because a very large percentage of the connected objects are not protected, or not enough.

Most attacks demonstrated on connected objects are based on software flaws and use programming techniques to infect a system. Even small objects such as smart light bulbs [2] or funny baby phones such as Karotz, hacked in 2013 among many other devices [3], have been shown to be prone to attacks. Such attacks can lead to denial of service, privacy violations, or can allow a hacker to use these devices to develop other attacks. In some cases, blocking a service or threatening the user can also aim at getting a ransom to be paid. In consequence, every connected object requires some attention to security constraints, of course at a level depending on the criticality of the application and of the managed data; taking control of a light bulb does not have the same immediate consequences as taking control of a running car, as demonstrated in [10], or medical devices such as insulin pumps [11] or pacemakers, as announced early at the Ruxcon Breakpoint conference in 2012. Consequences can however be on a large scale in many cases (not only for simple light bulbs), as illustrated by the recent recall of 465,000 pacemakers by the Food and Drug Administration in the US [12]. Cost constraints are also an important parameter; security approaches cannot have the same cost for a car and a light bulb.

In the typical area of secured applications, smart cards are well protected but most often use specific and well-controlled software; this is not the case for most connected objects. In addition, the certification procedures used in the smart card world are not adapted to the IoT constraints in terms of cost and time-to-market implications. However, well known attacks in the smart card world, called "hardware attacks", can also be applied to IoT objects, in addition to software attacks. Such attacks can be based on so-called "side channels", i.e., recording physical characteristics such as execution time, power consumption, or electromagnetic emissions [13]. They can also be based on controlled perturbations of the computations [5]. Such attacks can be used in order to recover a secret encryption key used to protect the software execution and modification (requiring e.g., authentication to update a firmware) or data (by encrypting it before transmission). Some attacks require very sophisticated equipment and skills, but some can easily be mounted at low cost. We will show one example in the next section.

The example of the smart lamps in [14] is interesting because the global attack involved a first hardware attack (here, a Correlation Power Analysis, or CPA, attack) in order to extract the master secret key for each type of lamp; then the key was used to propagate a worm through Zigbee connections, the modified firmware being authenticated with the extracted key. With this combination of hardware and software attack, the researchers demonstrated their capability to take control of a very large number of lamps in a whole city area, making for example feasible a massive DDoS (Distributed Deny of Service) attack even making use of a drone to extend the attack area.

Stolen or misused critical data is another threat and as already mentioned in the introduction the problem is today increasingly related to outsourcing in the cloud. Even if data is sent encrypted for external storage, it has in general to be decrypted by the service provider if some computations have also to be outsourced, and not only storage. This means that the private keys are available to the service provider who has to be trusted and to ensure that its infrastructures are perfectly robust against hackers. To avoid this, homomorphic encryption can be used. In that case, computations can be made directly on encrypted data, using various schemes [4]. However, such encryption schemes are extremely costly, both in terms of performance and the size of the data. This approach is therefore not suited to protect data in simple objects but may be a solution for some very critical applications. It has nevertheless to be optimized in order to make the overheads more acceptable; this implies algorithm improvements, as seen in the last few years, but also implementation improvements including hardware acceleration of the most expensive computations.

We will focus in the next section on one of the easiest hardware attacks a hacker can take advantage of and we will show that efficient protection can be achieved at very low cost.

Then we will show that hardware acceleration can be achieved with a lot of flexibility in the case of homomorphic encryption.

3. Countering Simple Power Analysis during Cryptographic Computations

3.1. Some More Background on Hardware-Based Attacks and Countermeasures

The goal of this paper is not to explain in detail hardware attacks or some cryptographic schemes. However, some background is necessary to understand the problem we will address and the proposed approach.

Many protocols for authentication or key exchanges are based on asymmetric cryptography. With such algorithms, a public key can be distributed to many people, while another key is kept secret. RSA [15] is widely used as an asymmetric cryptographic scheme. However, it is today increasingly being replaced by Elliptic Curve Cryptography (ECC) [16,17]. Since the underlying mathematical problem is more complex, ECC can be used with keys much shorter than RSA for a given security level.

From a purely mathematical point of view, the algorithms are robust. However, naïve implementation can lead to very weak systems. In the case of RSA, multiplications depend on the value of the secret key bits. In case of a naïve implementation, it is very easy to identify when multiplications are performed, just analyzing a single power consumption trace recorded during the computation, as different computed operations show different consumption patterns. In consequence, this single measurement can leak the whole secret key, no matter the number of bits. This is called a Simple Power Analysis (SPA) attack. For ECC, the same basic problem exists in the case of the classic Weierstrass elliptic curves. In that case, the operations depending on the secret key are the so-called "point additions". Without detailing the exact operations to be performed, which are much more complex than a classical integer addition, this step is also very easy to identify on a power consumption trace because it requires many commutations of gates. Figure 1 illustrates the power consumption of a naïve implementation, when starting an ECC encryption. This measurement was taken on the power pins of the Kintex-7 FPGA implemented on a Sakura-X board, as will be explained with more details in Section 3.3 when presenting our results. The critical step for both performance and security is the "scalar multiplication", i.e., the multiplication of a point P of the chosen elliptic curve with a scalar k. Figure 2 summarizes the simplest algorithm to compute a scalar multiplication, called "Double and Add", with left to right computations. At each iteration of the computation loop, either only a "point doubling" is made, or a "point addition" is also made after, depending on the value of the scalar. The two basic operations are quite different for Weierstrass curves, so the power consumption patterns are easy to identify, leaking most of the secret key bits. Some bits may not be directly identified, mostly when starting the measurement, due to some filtering in the circuit that partially compensates the quick increase in power requirements as seen from the external pins of the circuit. Also, quick variations are not necessarily well recorded, especially when low-cost equipment with a small bandwidth is used for current measurement. However, these missing bits can easily be recovered through a brute force attack, i.e., trying all remaining possibilities.

The results that we will comment in the next sections are focused on a scalar multiplication implemented with this simple algorithm. More optimized algorithms exist, but have a significant impact on computation time and/or power consumption. As a consequence, they cannot always be used in constrained implementations e.g., for the IoT.

Protections (or so-called countermeasures) exist, either at the algorithmic, programming or hardware implementation level. The most common consists in systematically carrying out both operations at each iteration, even when the second one is useless. The algorithm is called "Double and Add Always". Such a protection can still be identified because the useless operation results cannot be recorded, at least not in the same register or memory address as useful ones. However, the analysis becomes much more difficult and requires more sophisticated attacks such as Differential Power Analysis (DPA) or Correlation Power Analysis (CPA), or similar attacks based on recording electromagnetic emissions with more local information in the measurements. Much more advanced skills and equipment are required, so this is not affordable for any hacker. The main problem of such a countermeasure is, however, the loss in performance. In the IoT context, with systems designed to run at the lowest possible cost and with little power consumption, the available computing power is in general relatively small. Making many useless complex operations has therefore a noticeable impact on the global system features.

When lightweight implementations are sought, the usual countermeasures are therefore not a good answer. Other solutions must be proposed, allowing at least a protection against the simplest attacks (SPA).

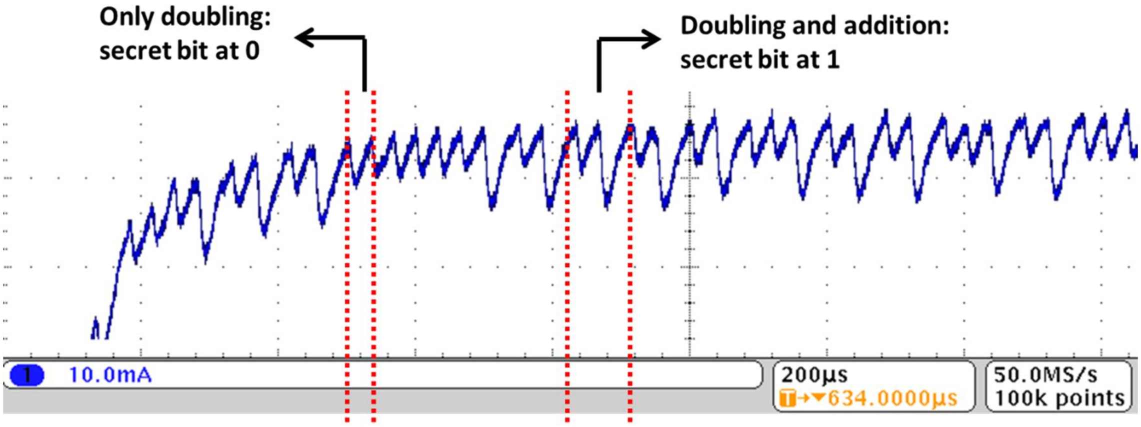
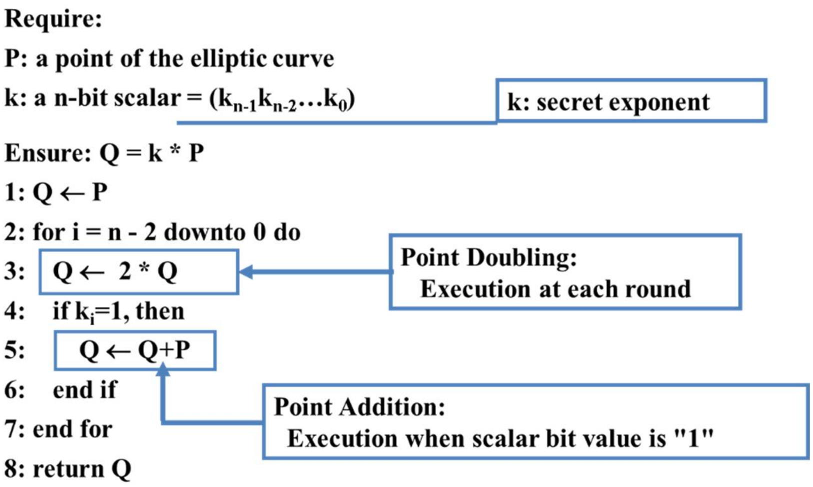


Figure 1. Example of power consumption trace for a circuit with a naïve implementation of Elliptic Curve Cryptography (ECC).



```
Require:
P: a point of the elliptic curve
k: a n-bit scalar = $(k_{n-1}k_{n-2}\ldots k_0)$
Ensure: Q = k * P
Q ← P
for i = n - 2 downto 0 do
    Q ← 2 * Q
    if $k_i$=1, then
        Q ← Q+P
    end if
end for
return Q
```

Figure 2. The most usual "Double and Add" algorithm for scalar multiplication.

3.2. Proposed Approach: Basic Idea

We assume here a very classic implementation of the system, making use of a general purpose microprocessor and a hardware accelerator to perform the most complex cryptographic computations. For example, in the case of ECC, the first operation to be implemented in hardware both for acceleration and power reduction is the scalar multiplication, as introduced in Section 3.1. The scalar value is most often the secret that the hacker wants to discover. During the computations performed by the hardware accelerator, the main processor is in general halted, waiting for the result. We also assume that the processor and the hardware accelerator are implemented in the same circuit (e.g., on a FPGA or in a MCU) and the hacker only has access to the external power pins, and can therefore only record the global power consumption.

Our goal is to avoid implementing countermeasures directly in the cryptographic accelerator, while protecting the whole system against the simplest attacks. The idea is therefore to use computations on the main processor in order to perturb power consumption measurements during critical computations of the cryptographic accelerator. Since only the global power consumption can be recorded, the proposed protection consists in adding random variations in the power trace so that it is no longer possible to identify the critical computation steps.

This approach was proposed in [18] for a particular example, but it is completely generic and can be used with both hard processor cores (e.g., an ARM processor in a Xilinx FPGA) or synthesized soft cores (e.g., a NIOS processor in a Altera FPGA). We will here demonstrate the feasibility in one particular case based on the Microblaze soft core from Xilinx, but any other processor may be used in the same way.

In order to randomize the power variations, a random number generator has to be used. Most systems with cryptographic computations have such a generator available, often a True Random Number Generator (TRNG). A small part of the generated random bits can be used to control our countermeasure. In case such a generator is not already available, a simple pseudo-random generator can be implemented at very low cost. In our case, we used for example a 32-bit Linear Feedback Shift Register (LFSR), requiring only 32 flip-flops and a few gates but generating a sequence of about 4 gigabits, sufficient to avoid any repetition and therefore allow the identification of the sequence in a reasonable time when the attack is not performed by highly skilled experts with strong equipment.

3.3. Proposed Approach: General Purpose Processor Characterization

In order to perturb the measurements made by the hacker, two main parameters have to be taken into account. First, the power consumption variations must be random and if possible not reproducible. In addition to the generation of random bits, it is therefore necessary to partition the software run on the main processor into relatively small functions with different power consumption profiles. The sequence of functions run during a given cryptographic computation is selected using the random bits. Second, the power consumption induced by each of these functions must be sufficient to noticeably modify the measures. It is therefore necessary to identify processor instructions inducing sufficient current, but not necessarily the maximum current in order to keep the total additional power consumption as small as possible.

A first step to prepare the proposed countermeasure is therefore to characterize the power consumption of the processor instructions. This has to be done one time for each new processor to be used and/or each new implementation technology. As a matter of fact, this characterization depends on the Instruction Set Architecture (ISA) of the selected processor, but also on the device on which the processor is implemented. In our case, we have characterized each instruction in the ISA using very simple loops, executing 1000 NOPs ("No-operation" instruction), then 1000 times the selected instruction, then again 1000 NOPs. Of course, the number of executions may be different without much impact on the results. The exact implementation of the NOP depends on the ISA—as an example, for the processor used during our experiments, it corresponded to an instruction “xor %0, %0, %0”—thus the power consumption is actually not null, and potentially not minimal. However, Figure 3 illustrates the usual form of the records obtained on the oscilloscope for such executions. The difference between the NOP consumption level and the consumption when repeating a different instruction is clearly visible and the characterization of the ISA can thus be automated. The loop control (execution of branch instructions) is similar in all cases, so it just induces some noise, a non-significant perturbation when comparing the power consumption between instructions. All the measures presented in this paper were obtained using the specific power-consumption measurement facilities of the Sakura-X board, connected to a Hall-effect probe combined with a 5-spire connection, and a digital oscilloscope.

In practice, the characterization of each instruction depends not only on the addressing mode, but also on the data used during the computations.

The general idea presented in the previous section can therefore be improved by also selecting randomly the data processed during the execution of the instruction sequences used for our countermeasure.

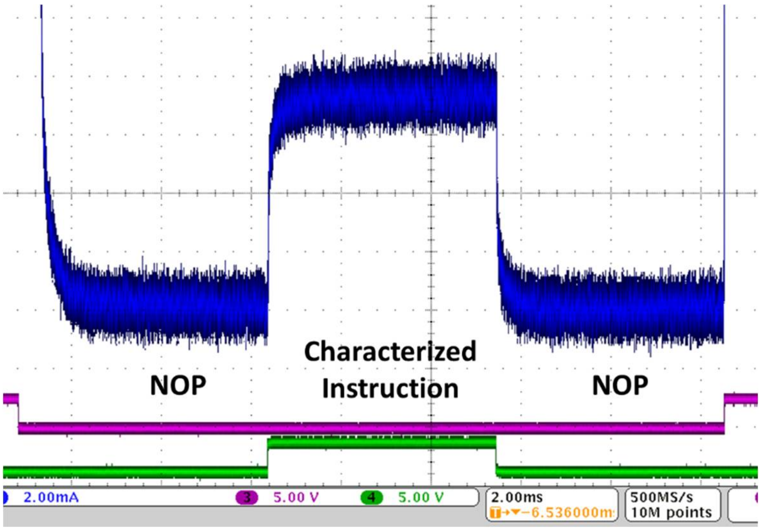


Figure 3. Example of power measurement for instruction characterization.

3.4. Automated SPA

As explained in [18], a SPA attack has been automated using Matlab in order to demonstrate (1) that the naïve implementation can easily be broken, and (2) the protected implementation cannot be broken in the same way. Figure 4 illustrates the principle used to discriminate point doublings and point additions during a scalar product computation. A similar approach may be used e.g., to discriminate operations during a RSA computation. Basically, thresholds are defined to identify which operation was carried out when there is a peak in the current trace. Figure 4 shows that different thresholds can be used with similar results since the power consumption is very different from one operation to the other. The Matlab input file is directly generated from the oscilloscope records. The upper level (green) is the average of the peaks, the lower level (black) is the middle of the peak values and the intermediate level (red) is the average of the two others. In all cases, operations can be efficiently discriminated. A partitioning of the record on a time basis was not used because each operation (point doubling or point addition) can have slightly different durations depending on the data.

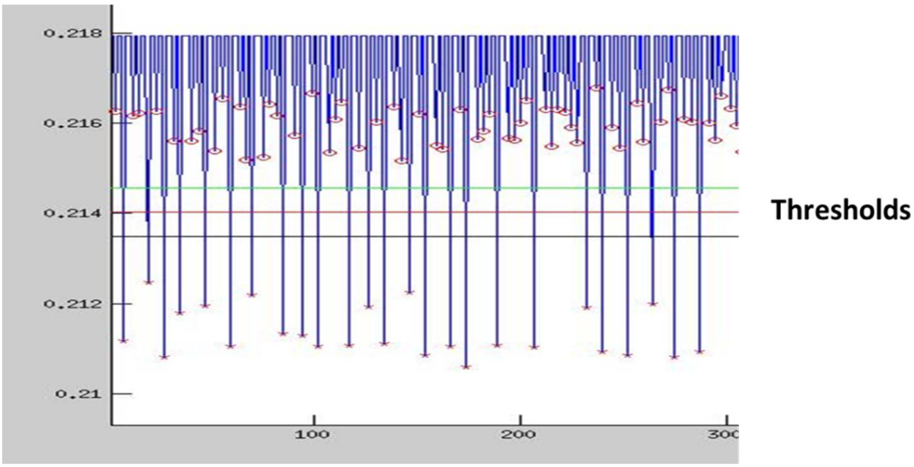


Figure 4. Automatic operation identification under Matlab.

3.5. Results Obtained from a Case Study

The system used for experimentation was based, as already mentioned, on a Microblaze general purpose processor soft-core implemented in the Xilinx Kintex-7 FPGA on a Sakura-X board, along with an accelerator for ECC scalar multiplication. The accelerator performs the scalar product operations without any countermeasure against side channel analysis. A very simple pseudo-random number generator (PRNG) was used to select the sequence of functions run on the main processor when activating the countermeasure. For practical use, a more efficient generator chosen among the various True Random Number Generator (TRNG) structures available in the literature may be chosen. Such structures would mainly have an impact on the non-deterministic and repeatability characteristic of the measurements, but would not significantly change the conclusions for short attacks.

The power consumption recorded during the computation is shown in Figure 1. The scalar product computations were carried out using 256-bit keys. The automated SPA attack on such records could recover about 240 bits from a single measurement.

The characterization of the Microblaze processor instructions led to some surprises. The average current consumption of the floating point instructions was found to be quite small compared to simpler instructions. It was the same for example for integer multiplications. This is probably due to their implementation in this processor, inducing long execution times. On the other hand, this showed that complex instructions are not necessarily those inducing the largest perturbations in the current measurement. The immediate addressing mode was found to lead to the highest level of power consumption. As illustrated in Table 1, very large discrepancies were recorded between instructions and many did actually consume less on average than the NOP. Even in a given category, instructions can have very different characteristics (e.g., for integer computations, the first two lines in Table 1). Also, the two results reported for addic illustrate the impact of the data used.

Table 1. Examples of instruction characterizations.

Instruction	Difference to NOP
rsubic %0, %0:1000, 5	+11.76 mA
mulh %0, %0:1000, %1:−2	−7.6 mA
addic %0, %0:1000, 1	+1.63 mA
addic %0, %0:1000, −20	+3.28 mA
xori %0, %0:1000, −20	+11.24 mA
bsrli %0, %0:1000, 5	−5.04 mA
flt %0:5.264, %1:15	−5.08 mA
lhu %0, %0:1000, %1:−2	−9.32 mA

Memory access instructions have large discrepancies in power consumption, depending on the address used. Also, such instructions have to be used with care due to border-side effects.

Finally, we selected for the Microblaze a set of 8 functions based on various instructions [18]. Each function had an execution time of the same order of magnitude as the basic cryptographic functions (point doubling and point addition), but all functions did not have exactly the same duration, leading to unpredictable overlaps in time, depending on the random selection of the function sequences. Randomness of data still increases the variations in terms of time overlaps. Depending on the function, the average current consumed by the general purpose processor was between 4 and 9 mA, while the average current variations were between 1 and 2 mA for the cryptographic core during the scalar product computations.

As mentioned in Section 3.3, the implemented countermeasure involved both a random selection of selected small functions to be run on the general purpose processor and a random modification of the data processed by these functions. The 32 bits of our PRNG were therefore used in part to select the next function to be executed (3 bits were sufficient since we had 8 functions) and the remaining bits were used to generate the seeds of manipulated data.

When adding the proposed countermeasure, the measurement with the same experimental set-up as used for Figure 1 is shown in Figure 5. It is clearly no longer possible to identify the sequence of point doubling and addition operations at first glance. Moreover, the automated attack presented in the previous section is no longer able to identify the secret key bits.

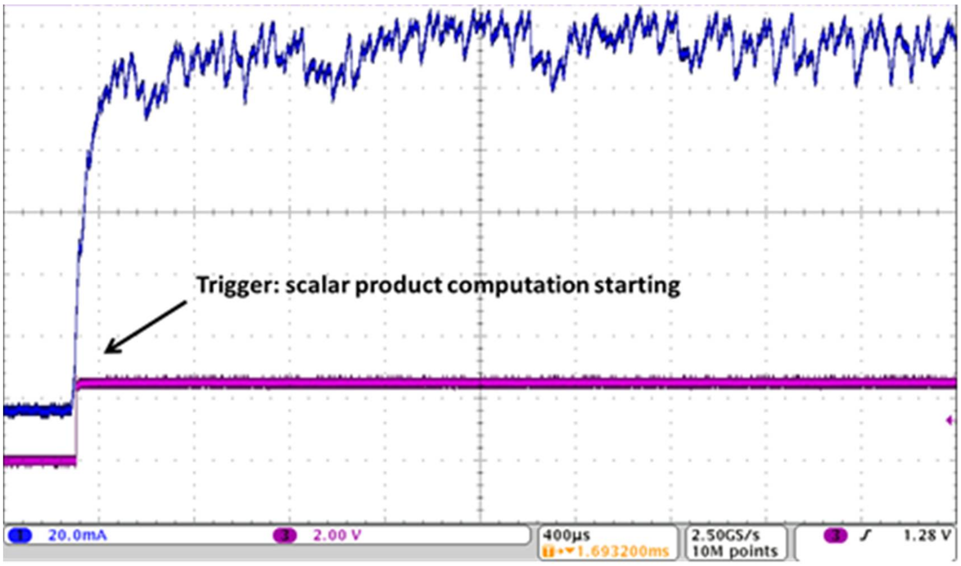


Figure 5. Effect of the countermeasure: power trace as recorded on the pins of the Kintex-7 FPGA after countermeasure activation, to be compared to the trace in Figure 1.

In our implementation, the increase in average current during the scalar product computation was found to be +16% (the average current increase was 7 mA with respect to the unprotected version having the main processor in polling mode). However, in terms of performance and energy our proposal is more efficient than the usual "Double and Add Always" countermeasure, consisting of systematically computing an addition after a doubling. First, our countermeasure does not imply any performance loss since no dummy operation is added to the useful computation. In consequence, the additional energy consumed by the countermeasure only corresponds to the +16% current during the nominal computation time. Using the "Double and Add Always" approach, the computation duration is increased on an average by 40%, considering the time required by the two basic operations for our implementation and an average +50% of additions required for the countermeasure (the exact increase of course depends on the number of 1s in the key). This leads also to an average 40% increase in energy consumption, even if the average current is not modified during the computation.

Combining the execution of specific software with the hardware operations is therefore an efficient way to implement countermeasures. No extra hardware is necessary (except the very simple random number generator, if one is not already available in the system) and the hardware cryptographic accelerator can remain as simple as possible for the selected operations. Overheads are limited to an increase in power consumption, due to the software execution, but as discussed the loss in energy is for our case study about half of that required by the most usual approach that also induces noticeable performance penalties.

4. Accelerating Homomorphic Encryption

4.1. Background and Acceleration Objectives

In the previous sections, hardware/software partitioning allowed us to increase the robustness with respect to some hardware attacks. In this section, we will show how hardware/software partitioning combined with High Level Synthesis (HLS) can be exploited to efficiently increase the performance and flexibility of security schemes. The case study is based on one of the most complex cryptographic approaches to date, i.e., homomorphic encryption, but not limited to a specific scheme.

As previously mentioned, homomorphic encryption is used to enable direct computations on encrypted data, preserving their confidentiality. Some schemes are partially homomorphic, i.e., they can compute only one type of operation (i.e., addition or multiplication) without decryption. RSA is for example homomorphic for multiplication. The Full Homomorphic Encryption (FHE), although envisioned in 1978 [19], had no practical demonstration during a long period. Since the breakthrough of Gentry's approach in 2009 [20], a lot of contributions have been proposed, leading to schemes such as FV or Yashe [6]. Such schemes based on the Ring Learning With Errors (RLWE) problem have become the most promising ones, even if a vulnerability has been found for Yashe. All these schemes, including more recent ones, need complex computations, mostly polynomial, on very large sets of data. As shown in Figure 6, the most expensive function is the polynomial multiplication, associated with the modular reduction. The Gaussian sampling then has a noticeable contribution, but other functions such as polynomial addition or integer operations are in comparison very fast. Polynomial multiplications require between 41% and 58% of the global execution time in software depending on the selected scheme; they are therefore the main target for hardware acceleration. We will thus focus in this paper on an efficient implementation of modular polynomial multiplications, no matter which exact FHE scheme is selected. Such multiplications have three main parameters: the degree of the polynomials, the size of the coefficients and an irreducible (cyclotomic) polynomial used for modular polynomial reduction. In order to be as general as possible in terms of the application and cryptographic scheme, the hardware acceleration must allow a designer to freely choose each of these parameters.

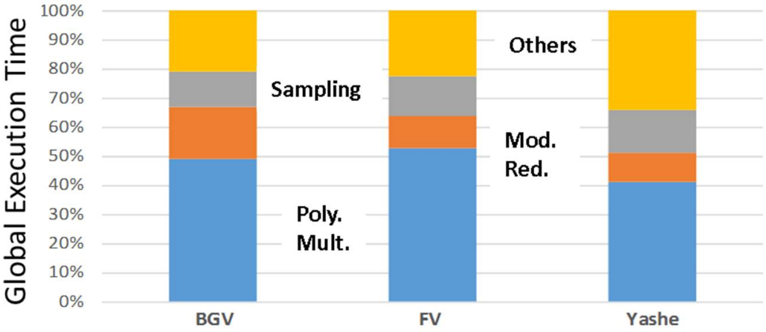


Figure 6. Software implementation profiling of the FHE schemes BGV [21], FV and Yashe [6], showing the impact on execution time of the polynomial multiplication, modular reduction, Gaussian sampling and all other functions.

Another point to be taken into account is the approach called "batching" [22]. This optimization allows simultaneous computations on several bits of the plaintext, thus improving both the total size of the encrypted data and the global computation time. However, it is not compatible with some polynomials used for modular reduction and especially the most usual ones of the form $f(x) = x^n + 1$. Allowing such optimizations therefore again requires providing acceleration without imposing restrictions on the polynomials that can be used for modular reduction.

4.2. Acceleration Approach: Principles

The polynomials used in FHE schemes have in general a very large degree and can also have large coefficients. The sizes are determined with respect to the specified security level and to the number of multiplications to be carried out successively on encrypted data for a given application (multiplicative depth). As reported for example in [6], the degree of the polynomials can easily be over 10,000 and each coefficient may have several hundreds of bits. Computations on data with such sizes require either expensive computers or very long times. We will not discuss here the choice of the parameters, but rather how to improve the computation times with low cost equipment without restrictions with respect to the selected FHE scheme.

In order to perform efficient multiplications on such large polynomials, we propose a four-level partitioning. Computations on the large polynomials are made in three steps: decomposing the polynomials into smaller ones, then computing the products of each couple of sub-polynomials, and finally reconstructing the global product. During the computations on the small polynomials, the products of the large-size coefficients are also performed in three steps: each large coefficient is converted into a set of smaller values by applying a RNS (Residue Number System) transformation [23], allowing us to perform parallel computations, then the partial products are made on the RNS basis, and finally the inverse transformation is applied. The four levels correspond therefore to computations on large/small (sub-)polynomials and large/small (sub-)coefficients. In the case of a small enough degree or coefficient size, one (or two) of the levels can be suppressed, i.e., polynomials and/or coefficients do not have to be decomposed, reducing the number of steps in the global computation. However, this four-level decomposition allows us to manage any degree and any coefficient size required by a given application or scheme.

The operations performed at each level for the modular polynomial multiplication are then:

- Level 1: decomposition of large degree polynomials, reconstruction, and modular reduction;
- Level 2: generation of the RNS basis used to decompose the coefficients;
- Level 3: polynomial multiplication;
- Level 4: RNS transformation, coefficient multiplication, modular reduction, and inverse RNS transformation.

All operations do not have the same complexity and therefore do not require the same computing power. In order to achieve the best trade-off between costs, computation length and flexibility, we propose to partition the set of operations between the software and hardware implementations. The hardware accelerator is dedicated to the most demanding computations and the other functions remain executed by a general-purpose microprocessor. This can be implemented on various FPGAs having an embedded processor hard core, on FPGAs with a synthesizable processor soft core, or even on a FPGA board used as a hardware accelerator and connected to a computer through e.g., a PCI-Express bus. All three solutions are feasible, with different trade-offs and constraints, taking care of the time required by the data transfers between the computations implemented in hardware and the executed software.

In our case, the partitioning was defined as follows. Levels 1 and 2 were assigned to software and levels 3 and 4 were implemented in hardware. Implementing all levels in hardware may have been possible but reduces flexibility and increases requirements on the hardware platform without a significant performance improvement.

Seeking flexibility, an important point is the applicability of the approach to many different hardware accelerators, both in terms of technology or platform provider (e.g., Xilinx or Altera FPGAs) and in terms of the complexity and cost of the selected platform. A large FPGA may allow for example more computations in parallel on the transformed coefficients but at a higher cost than smaller FPGAs. Our global approach does not depend on the selected trade-off that will only impact the generation of the RNS basis and the level of performance that can be achieved.

In addition to this global approach, an algorithm has to be chosen to perform the multiplications. Three types of algorithms have mainly been used in the literature: the simplest one called "schoolbook" [7], the Karatsuba algorithm [24] and the FFT/NTT e.g., [8]. FFT/NTT is the preferred algorithm in many works because it is asymptotically the most efficient in terms of complexity. However, this algorithm induces a lot of constraints and in particular is not compatible with batching, since the modular reduction must be made with $f(x) = x^n + 1$. Also, due to our decomposition scheme, the multiplications are made on relatively small sizes so the asymptotic efficiency is far from being reached. It must be mentioned also that the pre- and post-processing required by this approach are very costly both in terms of computation time and in terms of hardware resources in the case of hardware acceleration. As a consequence, only two algorithms were considered to be adapted to our objectives. In the case of relatively small sizes after decompositions, once again, the asymptotic efficiency is not reached for the Karatsuba algorithm and it was shown that more efficient computations can be achieved on hardware with the simpler algorithm, if efficiently optimized for the hardware target [7]. Another advantage is that no constraint is imposed on any parameter, while the Karatsuba algorithm has some limitations with respect to the degree of the polynomials. Our choice aims at the largest flexibility, without a noticeable practical impact on performance, and preserving any new evolution (and constraints) of the FHE schemes in the next years.

A last important point in our global approach is how to obtain an optimized hardware accelerator in a short period of time and how to easily and quickly adapt the parameters to new application constraints and/or new FHE schemes in the context of the very quick evolution of this area. The proposed answer is to use High Level Synthesis (HLS), i.e., to generate the synthesizable hardware description from (optimized) C or C++ source code. With this approach, it has been demonstrated that complex modifications of the hardware architecture can be obtained in a very short time by changing few lines of code in the source code and running again the HLS tool [25]. To our knowledge, all implementations of hardware accelerators for FHE reported in the literature have been designed at lower level (RTL—Register Transfer Level), for a very limited set of parameter values and often optimized for a specific hardware target. Such descriptions are very difficult to change in an optimized way to adapt the accelerator to other parameter sets and/or other hardware platforms.

4.3. Proposed Approach: Results for the Modular Polynomial Multiplier

We will present here some results obtained with the general approach proposed in the previous section. Some other results and comparisons with other published implementations can be found in [26]. Results were obtained for several types of hardware platforms, including in particular Spartan 6 and Virtex 7 in order to find some partial comparisons with previously published works. With our approach, changing the platform is very easy. Our main prototype was implemented on a Zybo board, with a Zynq-7000 component including a dual-core Cortex-A9 ARM processor (only one core was used) and programmable logic equivalent to an Artix-7 FPGA. Between the embedded processor and the programmable logic, coefficients of the operand and result polynomials were transmitted through the high-performance AXI buses HP0–HP3 and control data were sent through the general purpose AXI buses (GP0–GP3).

The HLS tool selected for our demonstration was Augh [27]. All versions of the accelerators were synthesized with it, taking advantage of various options to obtain different trade-offs between hardware resource exploitation and performance. In addition to the flexibility allowed by such a synthesis, it was shown that the loss of performance compared with hand-made designs can be noticeably compensated by a large gain in resources. This is not only due to HLS, but also to our design choices regarding e.g., the multiplication algorithm.

Figure 7 summarizes the parameters of some accelerators obtained with our approach, by comparison with published "hand-made" accelerators. Depending on the constraints of the application and of the FHE scheme, we were able to provide a very large set of accelerators. As illustrated by the presented results, we easily reached parameters (degree or coefficient size) that were not considered in previous works. We are also able to very quickly adapt an implementation with respect to new requirements.

Table 2 shows some of the obtained results in terms of resource requirements and computation performance, compared with some of the most up-to-date published results. As illustrated by these figures, our approach is very efficient in terms of hardware complexity, in addition to its flexibility. As illustrated by the comparison with [7], we were able to generate solutions oriented towards either performance or low-cost implementation. Compared with full software implementations, our solutions can lead to a significant optimization factor in terms of performance with respect to the software libraries (HElib and FLINT). Measures with these libraries on a computer with an Intel Core i7-4770 processor clocked at 3.4 GHz and equipped with 16 GB of RAM reported for the FLINT version 2.4.5 with Kronecker substitution algorithm (the fastest one) or the HElib implementation a computation time around 100 µs [7], i.e., more than ten times slower than what can be achieved with some of our accelerators.

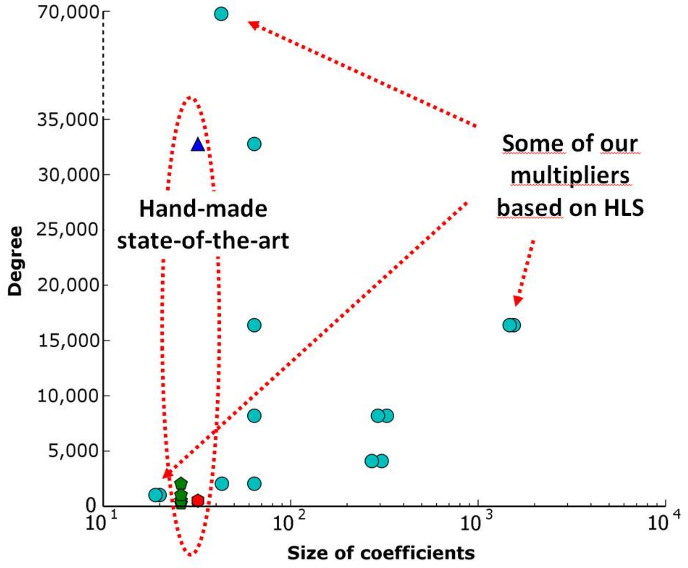


Figure 7. Flexibility with respect to published works.

Table 2. Examples of performance and complexity comparisons for polynomial multiplications.

Source	Degree	Coeff. Size	Platform	Slice LUT	Slice Register	DSP	Bram	Latency
[9]	1024	26	Spartan 6	10,801	3176	0	0	40.98 µs
This work	1024	26	Spartan 6	182	114	3	10	69.10 µs
[8]	32,768	32	Virtex 7	219,192	90,789	139	768	9.51 ms
This work	32,768	32	Virtex 7	13,568	7680	192	792	13.71 ms
[7]	512	32	Virtex 7	252,341	130,826	512	2048	4.11 µs
This work	512	32	Virtex 7	7032	920	368	0	5.27 µs
This work	512	32	Virtex 7	171	102	3	3	66.41 µs

Most computations can easily be parallelized, so a gain of a factor X in terms of hardware resources can easily be transformed into a similar gain in terms of computation performance, depending on the objectives of the system designer. Also, such cheap platforms can easily be run in parallel in contexts requiring many independent computations.

Due to our architecture and algorithmic choices, our implementations are also fully compatible with optimizations such as batching. Compared with other implementations, performance can therefore still be noticeably increased.

Gains in hardware resources have also a direct impact on the global power and energy consumption. It was shown in Section 3 that energy can be saved thanks to our approach. Here again, reducing the hardware complexity for a given level of performance leads to saved energy while achieving the same applicative objectives.

4.4. Proposed Approach: Generalization

The proposed approach is not limited to polynomial multiplication. As shown in Figure 6, the next most expensive function is Gaussian sampling. This function was also implemented in hardware using HLS and the results are shown in Table 3. Direct comparisons with previously published implementations are not possible, since we did not find references for this function. The results in the table illustrate mainly the flexibility of our approach in terms of resource requirements.

Table 3. Several implementations of Gaussian sampling on a Zybo board.

Degree	Coefficient Size	Slice LUT	Slice Register	DSP	Bram
256	493	230	720	3	15
256	7681	280	792	9	19
512	12,289	501	890	15	31

5. Conclusions

The results presented in this paper show that a good partitioning between hardware and software can be efficiently exploited to help improve performance and security in the IoT context. In the first case study, security was increased with respect to some hardware attacks, with very little impact compared to classical approaches. In the second case, the performance of complex encryption schemes was increased with respect to software implementations and hardware complexity was reduced in the case of hardware acceleration. A lot of flexibility was also made possible by combining hardware/software partitioning and high level synthesis. Further work will include the combination of both works to increase the robustness of homomorphic encryption against hardware attacks.

Author Contributions: R.L. partially conceived the experiments, contributed to and supervised the global work presented in this article, and is the main writer of the paper. A.M. is the main contributor to the work on homomorphic encryption acceleration. P.M. has co-supervised and contributed to the work on homomorphic encryption acceleration.

Acknowledgments: Work done by Asma Mkhinini has been partially supported by the French research project "SPICA" (FUI) and a grant from the University of Sousse (Tunisia). The authors wish to thank S. Pontié, T. Backenstrass and M. Blot for their experimental contributions on the protected ECC implementation and evaluation. We also thank the reviewers for their helpful comments.

Conflicts of Interest: The authors declare no conflict of interest. The founding sponsors had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, and in the decision to publish the results.

References

1. Internet of Things Research Study, Hewlett Packard Entreprise. 2015. Available online: http://files.asset.microfocus.com/4aa5-4759/en/4aa5-4759.pdf (accessed on 30 November 2017).
2. Ronen, E.; Shamir, A. Extended functionality attacks on IoT devices: The case of smart lights. In Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P), Saarbrücken, Germany, 21–24 March 2016; pp. 3–12.
3. Olawumi, O.; Väänänen, A.; Haataja, K.; Toivanen, P. Security issues in smart home and mobile health system: Threat analysis, possible countermeasures and lessons learned. *Int. J. Inf. Technol. Secur.* **2017**, *9*, 31–52.
4. Moore, C.; O'Neill, M.; O'Sullivan, E.; Doröz, Y.; Sunar, B. Practical homomorphic encryption: A survey. In Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), Melbourne, VIC, Australia, 1–5 June 2014; pp. 2792–2795.
5. Bar El, H.; Choukri, H.; Naccache, D.; Tunstall, M.; Whelan, C. The sorcerer's apprentice guide to fault attacks. *Proc. IEEE* **2006**, *94*, 370–382. [CrossRef]
6. Lepoint, T.; Naehrig, M. A comparison of the homomorphic encryption schemes FV and YASHE. In Proceedings of the 7th International Conference on Cryptology in Africa, Progress in Cryptology—AFRICACRYPT 2014, Marrakesh, Morocco, 28–30 May 2014; pp. 318–335.
7. Jayet-Griffon, C.; Cornelie, M.-A.; Maistri, P.; Elbaz-Vincent, P.H.; Leveugle, R. Polynomial Multipliers for Fully Homomorphic Encryption on FPGA. In Proceedings of the 2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig), Mexico City, Mexico, 7–9 December 2015; pp. 1–6.

8. Doröz, Y.; Öztürk, E.; Savas, E.; Sunar, B. Accelerating LTV Based Homomorphic Encryption in Reconfigurable Hardware. In *Cryptographic Hardware and Embedded Systems—CHES 2015*; Güneysu, T., Handschuh, H., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2015; Volume 9293.
9. Chen, D.D.; Mentens, N.; Vercauteren, F.; Roy, S.; Cheung, R.; Pao, D.; Verbauwhede, I. High-Speed Polynomial Multiplication Architecture for Ring-LWE and SHE Cryptosystems. *IEEE Trans. Circ. Syst.* **2015**, *62*, 157–166.
10. Black Hat USA 2015: The Full Story of How That Jeep Was Hacked. Available online: https://www.kaspersky.com/blog/blackhat-jeep-cherokee-hack-explained/9493/ (accessed on 21 November 2017).
11. Lethal Medical Device Hack Taken to Next Level. Available online: https://www.cso.com.au/article/404909/lethal_medical_device_hack_taken_next_level (accessed on 21 November 2017).
12. Firmware Update to Address Cybersecurity Vulnerabilities Identified in Abbott's (Formerly St. Jude Medical's) Implantable Cardiac Pacemakers: FDA Safety Communication. Available online: https://www.fda.gov/medicaldevices/safety/alertsandnotices/ucm573669.htm (accessed on 21 November 2017).
13. Koeune, F.; Standaert, F.X. A Tutorial on Physical Security and Side-Channel Attacks. In *Foundations of Security Analysis and Design III*; Aldini, A., Gorrieri, R., Martinelli, F., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2005; Volume 3655.
14. Ronen, E.; Shamir, A.; Achi-Or Weingarten, A.; O'Flynn, C. IoT goes nuclear: Creating a ZigBee chain reaction. In Proceedings of the IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 22–24 May 2017; pp. 195–212.
15. Rivest, R.L.; Shamir, A.; Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* **1978**, *21*, 120–126. [CrossRef]
16. Miller, V. Use of elliptic curves in cryptography. In Proceedings of the Advances in Cryptology (CRYPTO), Santa Barbara, CA, USA, 11–15 August 1986; pp. 417–426.
17. Koblitz, N. Elliptic curve cryptosystems. *Math. Comput.* **1987**, *48*, 203–209. [CrossRef]
18. Backenstrass, T.; Blot, M.; Pontié, S.; Leveugle, R. Protection of ECC computations against Side-Channel Attacks for lightweight implementations. In Proceedings of the 1st IEEE International Verification and Security Workshop, Sant Feliu de Guixols, Catalunya, Spain, 4–6 July 2016; pp. 2–7.
19. Rivest, R.L.; Adleman, L.; Dertouzos, M.L. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*; Academia Press: New York, NY, USA, 1978; pp. 169–179.
20. Gentry, C. A Fully Homomorphic Encryption Scheme. Ph.D. Dissertation, Stanford University, Stanford, CA, USA, 2009.
21. Brakerski, Z.; Gentry, C.; Vaikuntanathan, V. (Leveled) fully homomorphic encryption without bootstrapping. In Proceedings of the ACM 3rd Innovations in Theoretical Computer Science Conference (ITCS), Cambridge, MA, USA, 8–10 January 2012; pp. 309–325.
22. Smart, N.P.; Vercauteren, F. Fully homomorphic encryption with relatively small key and ciphertext sizes. In Proceedings of the 13th international conference on Practice and Theory in Public Key Cryptography (PKC'10), Paris, France, 26–28 May 2010; pp. 420–443.
23. Garner, H.L. The residue number system. *IRE Trans. Electron. Comput.* **1959**, *EC-8*, 140–147. [CrossRef]
24. Gathen, J.; Gerhard, J. *Modern Computer Algebra*, 3rd ed.; Cambridge University Press: New York, NY, USA, 2013.
25. Pontié, S.; Bourge, A.; Prost-Boucle, A.; Maistri, P.; Muller, O.; Leveugle, R.; Rousseau, F. HLS-based methodology for fast iterative development applied to Elliptic Curve arithmetic. In Proceedings of the Euromicro/IEEE Conference on Digital System Design (DSD), Limassol, Cyprus, 31 August–2 September 2016; pp. 511–518.
26. Mkhinini, A.; Maistri, P.; Leveugle, R.; Tourki, R. HLS design of a hardware accelerator for homomorphic encryption. In Proceedings of the IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS), Dresden, Germany, 19–21 April 2017.
27. Prost-Boucle, A. Augh Project. TIMA Laboratory, 2016. Available online: http://tima.imag.fr/sls/research-projects/augh/ (accessed on 30 November 2017).