# Fast polynomial arithmetic for Somewhat Homomorphic Encryption operations in hardware with Karatsuba algorithm

***Abstract*—Most practical *Somewhat Homomorphic Encryption* (SHE) schemes require the implementation of fast polynomial arithmetic in the ring $\mathbb{Z}_q[X]/f(X)$, for a given modulus $q$ and an irreducible polynomial $f(X)$. That is why hardware accelerators usually target the FFT/NTT algorithm, which has the smallest complexity asymptotically. Unlike standard approaches, this paper proposes a Karatsuba-based accelerator. Karatsuba implementation requires 3 steps: Pre-recursions producing several sub-polynomials, a term by term multiplication of sub-polynomials, and post-computations to reconstruct the output polynomial. Compared to FFT/NTT, Karatsuba can address various size of polynomials, and is sufficiently flexible to be adapted to specific operations required by SHE schemes. In this paper, we propose a hardware/software co-design where several Karatsuba recursions are made in software, and the remaining ones plus the sub-polynomial multiplication are made in hardware. We provide 3 different hardware approaches: An area efficient approach with 3 Karatsuba recursions in hardware, an intermediate design with 4 recursions, and a performance-oriented one with 5 recursions. The study evaluates proposed hardware accelerators for 3 FPGA platforms, the SoCkit and the DE5-net platforms from Terasic, and the Catapult platform from Microsoft. The area efficient approach can evaluate a degree-2559 polynomial multiplication in 2.44 ms and a relinearization/key switching evaluation in 2.29 ms, with an important save of hardware resources compared to FFT/NTT implementations. Compared to [1], our lightweight approach saves 57% of ALM resources, 46% of registers, 99.95% of embedded memory and 30% of DSPs. For the performance-oriented design, the accelerator can evaluate a degree-2559 polynomial multiplication in 1.24 ms and a relinearization/key switching evaluation in 1.1 ms.**

## I. Introduction

Lattice-based cryptography is on the spot since it has been demonstrated by C. Gentry in 2009 that it can be used to construct a *Fully Homomorphic Encryption* (FHE) scheme. With the help of FHE, one can compute any small algorithms in the cipher domain. To reduce the size of operands, one can bound the number of achievable multiplications on one ciphertext, constructing a *Somewhat Homomorphic Encryption* (SHE) scheme. Most practical encryption schemes are based on related problem, like the approximate-Great Common Divisor (a-GCD) problem [2][3], NTRU problem [4][5] and Ring-Learning With Error (R-LWE) problem [6][7][8][9][10][11]. Due to the relative flexibility of NTRU and R-LWE, recent software and hardware implementations target these schemes. Based on a ring of polynomials, both software and hardware approaches implement fast polynomial arithmetic using the FFT/NTT algorithm [12]. Two operations are usually performed by hardware accelerators: the polynomial multiplication and the relinearization/key switching. However, because SHE security knowledge is still under investigation, many hardware accelerators have quite unsecured parameters, in particular the oldest ones [13][14]. In [1], a classical but optimized FFT implementation is presented for two parameter sets. for parameter set $n$ = 4096 and $\log_2 q$ = 124 bits, the proposed accelerator performs a polynomial multiplication in 1.96 ms and a key switching in 4.79 ms. For parameters $n$ = 16384 and $\log_2 q$ = 512 bits, the polynomial multiplication is performed in 27.88 ms and the key switching in 20.8 ms. Authors of [1] implemented 512 $\times$ 512 bits multipliers with a small modular reduction by selecting a Solinas prime modulus [15]. Due to the size of polynomials and coefficients, a cache is implemented to connect the external memory used to store intermediate coefficients. They also report a bottleneck due to large integer multiplication. That is why in [16] a pre-computation is performed on polynomials to reduce the size of coefficients. They split a ciphertext into a few polynomials by using the Chinese Reminder Theorem (CRT) on each coefficient. The overall architecture is based on an array of crypto-units, which gives some flexibility to process several residue polynomials in parallel. For parameters $n$ = 32768 and $\log_2 q$ = 1228 bits, their accelerator performs an homomorphic multiplication in 121 ms including 25 ms spent for CRT.

Unlike standard approaches, we investigate the use of Karatsuba algorithm instead of FFT/NTT. Karatsuba has already demonstrated its effectiveness for polynomial multiplication in $GF_2$ [17], in particular for elliptic curve cryptography, but has not been much investigated for the general case. Compared to FFT, Karatsuba can address various sizes of polynomials, and above all, can be cleverly adapted to relinearization/key switching as it will be presented in the following. In this paper, we demonstrate that for various sizes and parameters, Karatsuba algorithm can be a good alternative to FFT.

The main contributions of this work are as follows:

- A presentation of a hardware/software co-design polynomial arithmetic accelerator for SHE schemes based on Karatsuba algorithm.
- A complete study of 3 variants of the hardware accelerator, implementing 3, 4 and 5 Karatsuba recursions in hardware.
- A thorough study of the hardware implementation, providing hardware implementation results for different FPGAs.

This paper is organized as follows. Section II recaps some key information on SHE crypto-systems based on the arithmetic on a ring of polynomials. Section III details the proposed architecture, the evolution of the architecture to address more Karatsuba operations in hardware, as well as the evaluation of hardware resources for different FPGAs. Section IV draws some conclusions.

## II. Theoretical background

### A. Notation

In the following, a polynomial is represented with an uppercase and its coefficients with a lowercase. For polynomial $A$, $a_i$ represents its $i^{th}$ coefficient. A vector of polynomials is noted in bold. For vector $\mathbf{A}$, $\mathbf{A}[i]$ is the $i^{th}$ polynomial of the vector. For coefficient $a_i$ of polynomial $A$, $a_{i,(j..k)}$ corresponds to the binary truncation of $a_i$ between bits $j$ and $k$. This notation is extended to polynomial $A$ where $A_{(j..k)}$ is the sub-polynomial where the truncation is applied to each coefficient. All operations are performed on the ring of polynomials $\mathbb{Z}_q[X]/f(X)$, where $q$ is an integer and $f(X)$ a degree-$n$ irreducible polynomial. $q$ and $n$ are chosen for security requirements, and also depend on the number of homomorphic operations required.

### B. Karatsuba algorithm

Karatsuba algorithm is an improvement of the standard polynomial multiplication algorithm which reduces the number of sub-products.
Input polynomials $P$ and $Q$ of degree $n-1$ are split into two parts of equivalent size, that is to say $\frac{n}{2}$ coefficients. Let $P_H$ and $P_L$ be two polynomials composed respectively by the coefficients of highest degree of $P$ and lowest degree of $P$. By the same way, one constructs $Q_H$ and $Q_L$. Input polynomials are now expressed as $P = P_L + P_H x^{n/2}$ and $Q = Q_L + Q_H x^{n/2}$.
When multiplying $P \times Q$ using the standard approach, the resulting decomposition is given by:

$$\begin{aligned} A \times B &= (P_L + P_H x^{n/2})(Q_L + Q_H x^{n/2}) \\ &= P_L Q_L + (P_L Q_H + P_H Q_L) x^{n/2} + P_H Q_H x^n \end{aligned}$$

Karatsuba optimization is based on noticing that the middle factor $(P_L Q_H + P_H Q_L)$ can be cleverly computed by $(P_L + P_H)(Q_L + Q_H) - P_L Q_L - P_H Q_H$. As one can quote, $P_L Q_L$ and $P_H Q_H$ are already computed and so do not require additional multiplications.
At the end, Karatsuba requires 3 sub-polynomial multiplications instead of 4, at a cost of two pre-computations, namely $(P_H + P_L)$ and $(Q_H + Q_L)$, and two post-computations for the reconstruction of the middle factor. However, these pre- and post-computations are made of additions and subtractions only. To further reduce the number of sub-products, Karatsuba algorithm can be recursively applied on sub-polynomials.

TABLE I: Parameters for FV extracted from [18] satisfying a security level $\lambda$ of 80 bits, with $\omega$ set to 27 bits.

| $L$ | $\log_2 q$ | $n$ |
|---|---|---|
| 1 | 48 | 904 |
| 2 | 73 | 1428 |
| 3 | 98 | 1951 |
| 4 | 125 | 2515 |
| 5 | 152 | 3077 |
| 6 | 179 | 3636 |
| 7 | 207 | 4215 |
| 8 | 235 | 4792 |
| 9 | 264 | 5388 |
| 10 | 293 | 5982 |

Because each recursion of Karatsuba halves the size of sub-polynomials, Karatsuba can achieve polynomial multiplication of degree $2^r(p+1)-1$, where $r$ is the number of Karastuba recursions and $p$ the degree of the smallest sub-polynomial.

### C. Relinearization/Key switching evaluation

The relinearization/key switching operation is a central bottleneck of SHE schemes. During the homomorphic multiplication, a polynomial multiplication is performed between ciphertexts. This leads to a malformed ciphertext which cannot be further used for homomorphic evaluation. The transformation to a well-formed ciphertext is called relinearization or key switching, depending on the scheme. In both cases, the core arithmetic is the same. One has to select a parameter $\omega$, which is an adjustment variable in practice. The ciphertext is split into $l = \lceil \log_2 q / \log_2 \omega \rceil$ parts, and each sub-polynomial is multiplied/accumulated with an associated key $\gamma[i]$. The split operation is usually called $\text{WordDecomp}_{w,q}(A)$ :

- $\text{WordDecomp}_{w,q}(A)$ :
  $\mathbf{A} \in R_q^{l_{w,q}}$
  for $i$ in 0 to $l_{w,q} - 1$
    $l_0 = i \times \log_2 \omega$
    $l_1 = (i+1) \times \log_2 \omega - 1$
    $\mathbf{A}[i] = A_{(l_0..l_1)}$
  end for
  return $\mathbf{A}$

The formal writing of relinearization/key switching operation can be expressed as:

$$\mathbf{C} = \text{WordDecomp}_{w,q}(C)$$

$$\widetilde{C} = \sum_{i=1}^{l} \mathbf{C}[i]\gamma[i] \tag{1}$$

Where $C$ is the input ciphertext, $\gamma$ the vector of relinearization/key-switch keys and $\widetilde{C}$ the resulting ciphertext.

### D. Choosing parameters

In SHE, the size of operands is led by the number of operations performed on a given ciphertext. Each ciphertext has an initial noise which is growing between each homomorphic operation until making the decryption procedure faulty. To balance that issue, one has to increase the size of the modulus $q$, inferring an increase of the polynomial degree to maintain the required security level. Because the noise growth after an addition is much less important than after a multiplication, it is standard to determine parameters considering the number of required multiplications only. It is called multiplicative depth and is usually noted $L$. Table I gives some SHE parameters for FV scheme.

As it can be noticed, the degree $n$ of polynomial can be quite distant from a power of two. The FFT/NTT algorithm is much impacted by this issue, requiring to be below and as closed as possible to a power of two to be efficiently implemented. Two FFT/NTT are usually implemented, the *Positive Wrapped Convolution* (PWC) which is associated to the polynomial $x^n-1$, and the *Negative Wrapped Convolution* (NWC), which is associated to $x^n+1$. During the computation of the FFT/NTT-based multiplication, the resulting polynomial is reduced by $x^n-1$ for the PWC, and $x^n+1$ for the NWC. That is why in practice, one must double the size of the FFT/NTT for a basic multiplication. However, because $x^n+1$ is an irreducible polynomial, the NWC does not require to double the size of the FFT/NTT because homomorphic operations are in the ring $\mathbb{Z}_q[X]/f(X)$.

Even if the NWC is implemented most of the time for performance reason, in practice it brings important issues. Because $x^n+1$ is irreducible modulo 2, one cannot pack several messages into one ciphertext, disabling Single Instruction Multiple Data (SIMD) operations as explained in [19]. That implies large expansion of data compared to the plaintext. Because Karatsuba implements a standard multiplication, our approach does not have such a limitation.

To demonstrate the interest of Karatsuba approach, this paper focusses on multiplicative depth of 4, requiring polynomial arithmetic of degree-2515. For such parameters, a FFT/NTT of size 8192 is required in the standard approach, or 4096 for the NWC one. To be as close as possible to the required $n$, we set the smallest Karatsuba sub-polynomial to degree-4, associated with 9 Karatsuba recursions. That allows polynomial multiplications of degree at most $2^i p - 1 = (4+1)\cdot 2^9 - 1 = 2559$.

## III. Karatsuba implementation

The complete implementation relies on a hardware/software co-design approach. Figure 1 provides a high level flowchart for the proposed accelerator. On the software side, first pre-computations are performed in order to generate sub-polynomials. Then, sub-polynomials are sent continuously to the hardware accelerator which performs the remaining Karatsuba pre-computations. At that point, degree-4 sub-polynomials are generated in hardware. Then, sub-polynomials are multiplied using the standard multiplication algorithm. Following the polynomial multiplication, first post-computations are performed before sending back polynomials to the software. Finally, the software computes the remaining post-computations.

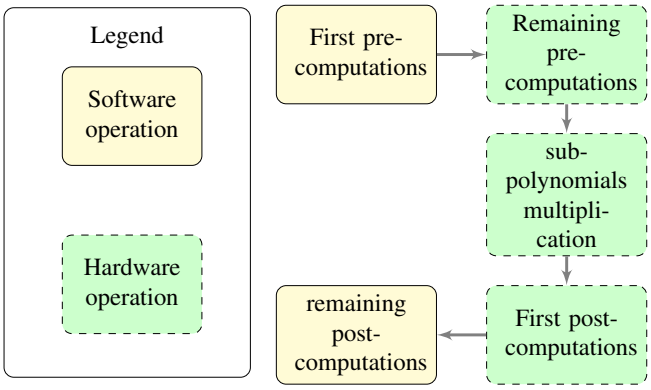


Fig. 1: Flowchart of the proposed software/hardware co-design approach.

In order to be able to send continuously polynomials to the hardware, proposed Karatsuba hardware implementation is a fully pipelined one. When input polynomial coefficients are sent in the ascending order, the accelerator produces output coefficients continuously after a few clock cycles, depending on the number of pipeline stages. Because the number of coefficients of output polynomials is greater than input ones, output polynomials are dispatched into two channels. The accelerator can run into two modes: A standard mode for the polynomial multiplication, and a relinearization/key switch mode for relinearization/key switching. For this second configuration, the relinearization key is required and must be sent at the same time than the polynomial to be relinearized/key switched.

To use embedded resources at their maximum frequency, we set the datapath to 27 bits. It is the consequence that embedded DSPs are efficiently implemented to run $27\times27$ bits integer products fastly. Because the size of coefficients is greater than 27 bits, namely 125 bits, each coefficient is split into 5 elements of 27 bits each. That implies that all elementary operations, including adders, subtractors and multipliers, are serialized. Another consequence is that our accelerator performs polynomial operations on coefficients of $27\times5$ = 135 bits instead of 125 bits required by the security parameters provided in Table I.

In the following, we propose 3 Karatsuba hardware implementations: a Karatsuba with 3 recursions in hardware called Karatsuba-3, which allows the multiplication of two degree-39 polynomials; an implementation with 4 recursions called Karatsuba-4, performing a degree-79 polynomial multiplication; and a version with 5 recursions in hardware called Karatsuba-5 for degree-159 polynomial multiplication.

### A. Karatsuba-3

Figure 2 provides the high-level view of the architecture of Karatsuba-3, where input sub-polynomials are named $P$ and $Q$. After the pre-computation and the pre-crossbar, the

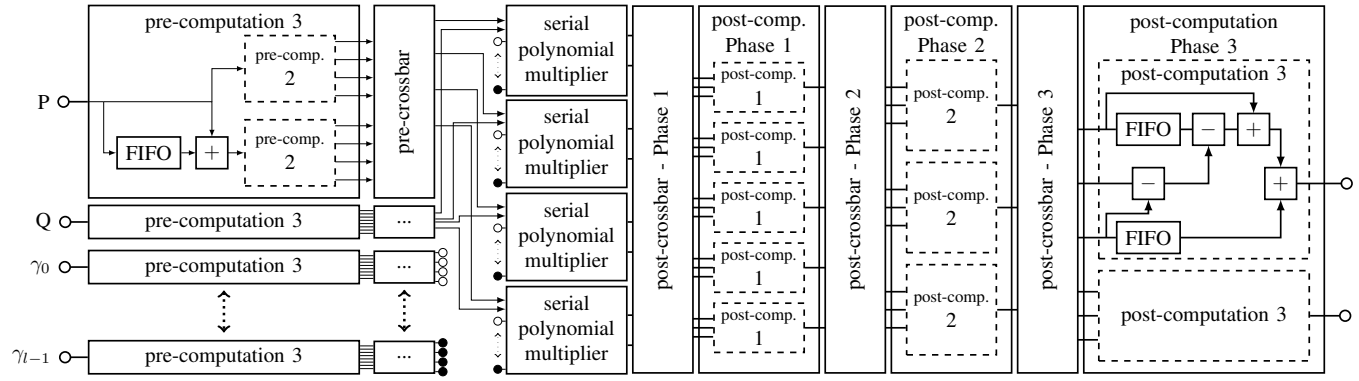



Fig. 2: Architecture of the Karatsuba hardware accelerator.

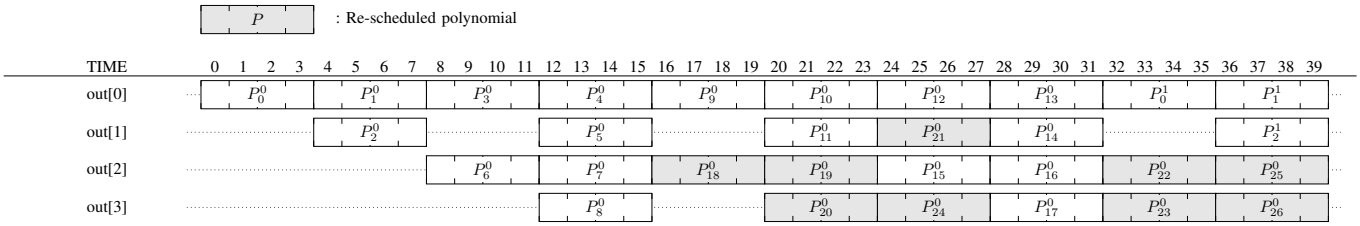
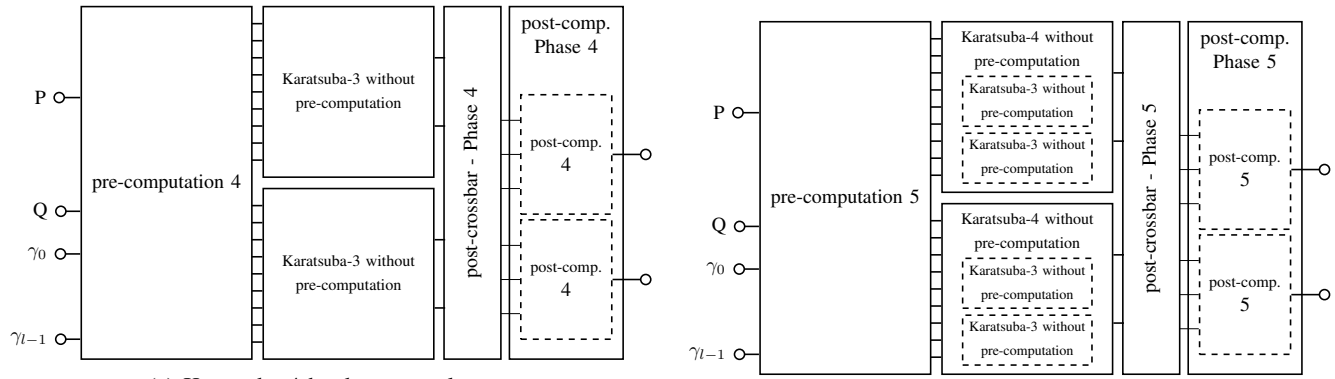



Fig. 3: Schedule example after a pre-computation with 3 recursions of Karatsuba.

accelerator generates 4 lines of sub-polynomials, which are multiplied in parallel. Post-crossbars and post-computations perform the reconstruction of the output polynomial.

*1) Pre-computation and pre-crossbar:* The pre-computation step is the first block of Karatsuba-3 and performs 3 pre-recursions of Karatsuba on the input polynomial. Our implementation is based on a recursive architecture as it can be seen in Figure 2. During a pre-computation, one has to split into two parts input polynomial $P$, namely $P_L$ and $P_H$. Because coefficients are sent in an ascending order, a FIFO is implemented in order to store first arrived coefficients, namely coefficients of $P_L$, before adding them term by term to last ones, namely coefficients of $P_H$. This creates two channels of polynomials: the first is just the input and produces $P_L$ and $P_H$; and the second one is just after the polynomial sum and produces sub-polynomial $P_L + P_H$ half of the time.
To perform an additional Karatsuba recursion, the same operation must be applied to $P_L$, $P_H$ and $P_L + P_H$. That is why the architecture is duplicated on each channel, with a minor modification on the FIFOs in order to manipulate polynomials of halve size, namely $P_L$, $P_H$ and $P_L$ + $P_H$.
Because each $P_L$ + $P_H$ channel produces sub-polynomials half of the time, after 3 pre-computations, several output channels haven't numerous valid sub-polynomials. That is why a pre-crossbar is implemented in order to re-schedule more efficiently sub-polynomials. Figure 3 proposes a scheduling for Karatsuba-3.

*2) Serial polynomial Multiplier and serial integer multiplier:* The serial polynomial Multiplier can run in two modes: a standard polynomial multiplier which requires polynomials $P$ and $Q$; and a relinearization/key switching mode which requires a polynomial $P$ and relinearization/key switching key $\gamma$. 4 serial polynomial multipliers are implemented because 4 channels of sub-polynomials are produced after the pre-crossbar of each input polynomial. As it can be seen in Figure 2, the first output of each pre-crossbar is connected to the first polynomial multiplier, the second output to the second multiplier, and so on. For readability reasons, connections for relinearization/key switching key are not represented, but mentioned with black and white circles.
Implemented degree-4 polynomial multipliers are based on the standard polynomial multiplication algorithm. In order to be able to send polynomials without interruption, a full-parallel design is implemented and requires 5 serial integer multipliers in parallel. By the same way, serial integer multipliers are based on the standard algorithm. Because polynomial coefficients are split in 5 segments of 27 bits each, the serial integer multiplier requires 5 DSPs. Thus, the serial polynomial multiplier requires 25 DSPs, and 100 for the overall accelerator.




(a) Karatsuba-4 hardware accelerator.

(b) Karatsuba-5 hardware accelerator.

Fig. 4: Proposed architectures to perform 4 and 5 Karatsuba recursions in hardware.

*3) Post-crossbar and post-computation:* A reverse scheduling is required to realign sub-polynomials before post-computations. However, if the post-crossbar is implemented following the same approach than the pre-crossbar, this one would be quite complex because all sub-polynomials would be aligned with the most delayed one. That is why in Figure 2, a successive implementation of partial post-crossbars and post-computations is implemented. Because the pre-crossbar only moved 1/3 of all sub-polynomials, 2/3 of sub-polynomials can be post-computed before any re-alignment, reducing the complexity of successive post-crossbars and also storage requirements. Unlike the pre-computation, because post-computations and post-crossbars are successively implemented, each post-computation unit is an elementary one, and so is not implemented using a recursive approach.

### *B. Implementing additional recursions in hardware*

The initial design implements 3 Karatsuba recursions in hardware and 6 recursions in software. In order to reduce software computation times, we propose an adaptation of Karatsuba-3 to execute further recursions in hardware. The study provides a method, especially for pre- and post-crossbars, and hardware results for 4 and 5 recursions.

*1) Impacts on the sub-systems:* Implementing additional recursions in hardware does not impact each sub-system equally. Due to the scalability of pre- and post-computation units, these components are not much impacted, but require additional logic and temporary storage. Degree-4 polynomial multipliers internal structure remains unchanged, because further Karatsuba recursions only means additional lanes of sub-polynomials at the output of pre-crossbars, and thus just more degree-4 polynomial multipliers in parallel. For pre- and post-crossbars, additional recursions means a more complex schedule of sub-polynomials, and thus possible complex crossbars. When 8 channels of sub-polynomials were produced for Karatsuba-3, 16 are produced for Karatsuba-4 and 32 for Karatsuba-5.

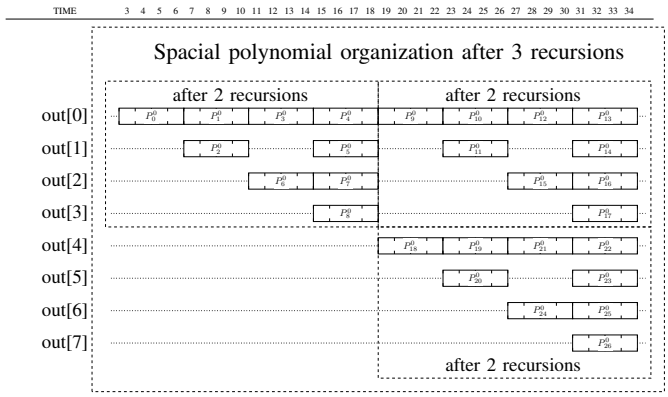



Fig. 5: Spacial organization of sub-polynomials after 3 Karatsuba pre-recursions, and the link with the spacial organization of sub-polynomials in the previous recursion.

*2) Adaptation of pre- and post-crossbars:* For 4 and 5 recursions in hardware, the pre-crossbar has to manage respectively 16 and 32 channels of sub-polynomials, which possibly requires a complex architecture when one wants to reduce as much as possible the number of channels in order to limit hardware resources consumption. Instead of designing complex crossbars, with a risk of reducing the design maximum frequency, we propose a method which does not require additional developments. Figure 5 presents the spacial organization of sub-polynomials after 3 Karatsuba pre-recursions, and the link with the spacial organization of sub-polynomials in the previous recursion. Because the pre-computation architecture is recursive, the spacial organization of sub-polynomials is recursive too. To determine the spacial organization of sub-polynomials for the next recursion, one

just has to duplicate the previous one two times: one time on the same channels, and the other time to fresh channels. Because the pre-crossbar role is to spatially organize in a better manner sub-polynomials, the previous remark on spacial organization of sub-polynomials between two successive Karatsuba recursions can be used to create a new somewhat efficient sub-polynomial schedule. To generate the Karatsuba pre-crossbar, one can reuse the Karatsuba-3 pre-schedule twice. One time on the first 8 channels of the pre-computation unit of Karatsuba-4, and the other one on the remaining channels. As one can see in Figure 5, the south-west area is free of sub-polynomials, which means that the second pre-crossbar will not run half of the time. That is why compared to the Karatsuba-3 schedule, DSPs usage only decreases by 25%. For 3 Karatsuba recursions, the DSPs usage is of 84.38%, it decreases to 63.29% for 4 recursions, and reaches 47.46% for 5 recursions.

To counterbalance, the post-crossbar for the $4^{th}$ iteration is very simple, because the spacial organization for the step has not been modified.

*3) High-level architectures:* Figures 4a and 4b provide the architecture of Karatsuba-4 and Karatsuba-5 designs. The Karatsuba-4 approach is presented in the following, but Karatsuba-5 design follows the same approach.

All pre-computations are generated in a unique entity, by using the proposed pre-computation unit presented in Section III-A1. Then, the Karatsuba-3 pre-crossbar must be applied twice: one time on the first 8 outputs of the pre-computation unit, and a second time on the 8 remaining outputs. Remaining computations on each branch until the $4^{th}$ post-computation remain the same than Karatsuba-3. Finally, the $4^{th}$ post-computation is performed after a quite simple post-crossbar.

As it can be noticed, the strategy provided to adapt pre- and post-crossbars leads to duplicate the logic 2 times for Karatsuba-4 compared to Karatsuba-3, and 4 times for Karatsuba-5. This remark leads to two main issues. First, because FPGAs have limited embedded multipliers, the number of DSPs required will drive the choice of the target FPGA. For 3 Karatsuba recursions, 100 DSPs are required, which is not a particular issue because even smaller FPGAs like a Cyclone V have such a number of multipliers. This is not all the time true for the next implementations, namely for 4 and 5 Karatsuba recursions, because respectively 200 and 400 DSPs are required. In particular, if one wants to target an embedded system with an energy efficient design, Cyclone V SoC family owns at most 112 DSPs, which limits the design to 3 recursions. Second, doubling the number of logic between each recursion will quickly lead to large design for small FPGAs. It will also limit the number of parallel Karatsuba implementable in parallel.

### C. Hardware implementations results

Table II provides hardware implementation results. In order to thoroughly explore the proposed approach, we address 3 different platforms. For embedded systems, we present results for a SoCkit platform from Terasic, embedding a small Cyclone V (5CSXFC6D6F31C6N) FPGA with 41,910 ALMs and 112 DSPs. The second platform is a DE5-net 450 from Terasic, especially designed for speeding up software applications with the help of a PCIe. This platform embeds a powerful Stratix V (5SGXEA7N2F45C2) FPGA with 234,720 ALMs and 256 DSPs. The last one is the Catapult platform from Microsoft, used in [1] to accelerate *Homomorphic Encryption*. With its 172,600 ALMs and above all its 1,590 DSPs, this platform can address all Karatsuba implementations. Due to the relatively low design frequency for the Cyclone V FPGA with the balanced mode of Quartus, we present implementation results when aggressive performance optimization mode is enabled in Quartus II. For Karatsuba-5 design, only the Catapult platform can embed this one due to the number of DSPs required, thus hardware results are only given for this platform. Our implementation is a fully pipelined implementation of Karatsuba, thus we do not provide number of clock cycles required for the accelerator, because it is not relevant. The overall computation time will be determined in Section III-D, when the complete design, including the software part running on the CPU will be presented. Hardware resources of the FFT/NTT implementation in [1] is also provided, able to perform a FFT/NTT of size 4096 with coefficients of size 125 bits.

*1) Karatsuba with 3 recursions:* As one can see, important efforts are required to reach acceptable performance for the Cyclone V approach. The balanced design runs at 148.7 MHz with 64% of logic consumption. If one wants to increase the maximum frequency, one can reach up to 184.4 MHz with minimum efforts, namely selecting the aggressive performance mode in Quartus II, but at a cost of 80% of the available logic. As one can see, the Quartus algorithm tries to limit the number of embedded memory to increase performances, at a cost of extra registers.

For Stratix V FPGAs, the design only requires 12.7% of logic resources for the DE5-net 450 platform, and 17.2% for the Catapult platform. Because the PCIe limits the maximum design frequency to 250 MHz, the frequency margin permits to implement other accelerator in parallel, or even other Karatsuba. For the DE5-net 450 platform, due to the limitation in terms of DSP, only 1 additional Karatsuba can be implemented. For the Catapult approach, a maximum of 5 Karatsuba can be implemented in parallel. However, in Altera FPGAs, the PCIe is implemented on the FPGA-side as a 256 bits bus, full duplex, implementing 5 Karatsuba-3 in parallel requires a bus of $27 \times 2 \times 5 = 270$ bits large. Thus, in practice, only 4 Karatsuba-3 can be implemented.

Compared to the FFT approach, our accelerator saves 57% of ALM resources, 46% of registers, 99.95% of embedded memory and about 30% of DSPs.

*2) Karatsuba with 4 recursions:* If one has a larger FPGA, the Karatsuba implementation with 4 recursions can be a good balance to accelerate SHE polynomial arithmetic. Karatsuba-4 requires 26.5% of the overall logic in the DE5-net 450 platform, and 36% for the Catapult one. Due to the limitation of DSPs on the DE5-net 450, no additional Karatsuba-4 design

TABLE II: Hardware implementations results compared to FFT implementation in [1].

| Algorithm | recursions | platform | mode | ALM | Registers | Embedded memory | DSP | $f_{max}$ |
|---|---|---|---|---|---|---|---|---|
| Karatsuba | 3 | SoCkit | Balanced | 27,034 (64.5%) | 70,273 | 12,855 | 100 (89%) | 148.7 MHz |
| | | | Performance | 33,915 (80.9%) | 93,310 | 4,297 | 100 (89%) | 184,4 MHz |
| Karatsuba | 3 | DE5-net | Balanced | 29,718 (12.7%) | 76,706 | 4,131 | 100 (39%) | 355.75 MHz |
| | | Catapult | Balanced | 29,715 (17.2%) | 76,730 | 4,131 | 100 (6.3%) | 334.0 MHz |
| | 4 | DE5-net | Balanced | 62,215 (26.5%) | 155,427 | 12,638 | 200 (78.1%) | 330.58 MHz |
| | | Catapult | Balanced | 62,202 (36%) | 155,299 | 12,638 | 200 (12.6%) | 355.75 MHz |
| | 5 | Catapult | Balanced | 124,978 (72.4%) | 309,416 | 101,380 | 400 (25.6%) | 321.34 MHz |
| FFT [1] | | Catapult | Performance | 69,058 (40%) | 144,747 | 8,031,568 | 144 (9.1%) | 100 MHz |

TABLE III: Implementations results compared to FFT implementation from [1], where (A) refers to the polynomial multiplication and (B) the relinearization/key switching.

| | | Our design | | | FFT [1] |
|---|---|---|---|---|---|
| Setup ($n$, $\log_2 q$) | | (2560, 125) | | | (4096, 125) |
| Hardware Karatsuba recursions | | 3 | 4 | 5 | |
| (A) | Software pre-computation | 531 µs | 385 µs | 270 µs | |
| | Hardware accelerator | 583.2 µs | 388.8 µs | 259.2 µs | |
| | Software post-computation | 1.35 ms | 931 µs | 710 µs | |
| | Total | 2.46 ms | 1.704 ms | 1.24 ms | 1.96 ms |
| (B) | Software pre-computation | 343 µs | 212 µs | 135 µs | |
| | Hardware accelerator | 583.2 µs | 388.8 µs | 259.2 µs | |
| | Software post-computation | 1.35 ms | 931 µs | 710 µs | |
| | Total | 2.28 ms | 1.53 ms | 1.104 ms | 4.79 ms |
| (A) + (B) | Total | 4.74 ms | 3.23 ms | 2.34 ms | 6.75 ms |
| Area × latency (ALM · s) | (A) | 73 | 105 | 154.9 | 136 |
| | (B) | 68 | 95 | 137 | 331 |
| | (A) + (B) | 141 | 201 | 292 | 466 |

can be implemented, and only 1 additional design for the Catapult one. Regarding the resources consumption, if we do not take into account embedded memory consumption, this design is the closest one in terms of resources consumption compared to the FFT implementation in [1].

*3) Karatsuba with 5 recursions:* For the Karatsuba with 5 recursions, one begins to reach the limit of the approach. First, only the Catapult platform can embed this accelerator due to the number of DSP required. Second, the number of logic resources does not allow the implementation of multiple units in parallel, because it requires 72.4% of the total amount of ALMs. However, this implementation reduces as much as possible the total computation time because the pre- and post-computations required in software are limited. As one can see, the number of embedded memory used grows up. A possible explanation is that each additional post-crossbar/post-computation requires to store even larger polynomials. For the $5^{th}$ recursion, a degree-159 polynomial must be stored, implying 21,600 bits. It is also possible that the tool tries to limit the use of registers, so ALMs, because the design begins to reach the device limits.

### D. Evaluation of the complete computation time

In order to completely evaluate degree-2559 polynomial multiplication and relinearization/key switch operation, the software computation of the remaining pre- and post-computations is required. Table III recaps software computation time for the 3 different hardware implementations, and the total computation time taking into account the hardware latency. It also provides the area-latency product for each design, taking into account the complete computation time (polynomial multiplication + Relin/key switch).

The software is executed on an Intel core i7-4910MQ with 4 cores running at 2.9 GHz. Hardware computation time is given by taking an operating frequency of 250 MHz, limitation due to the PCIe.

As one can see, our accelerator is very competitive compared to FFT/NTT implementation in terms of latency. Even with the Karatsuba-3 design, our accelerator outperforms FFT/NTT for the relinearization/Key switching step, with approximately 47% of computation time saved. This is a consequence that FFT/NTT has much less flexibility than Karatsuba approach. For a pure polynomial multiplication, the FFT/NTT still remains competitive for Karatsuba-3, due to the lower complexity asymptotically. However, this is not the

case when additional recursions are made in hardware, because the post-computation in software drops down. Thus, Karatsuba approach is a good alternative to FFT/NTT in any scenario.
The area - latency product provides a good indicator to compare designs, because it indicates the efficient use of FPGA resources. As one can see, the fully optimized Karatsuba-3 design has the lowest area - latency product compared to other Karatsuba implementations. It is a consequence of the not optimized schedule implemented for Karatsuba-4 and Karatsuba-5. For the polynomial multiplication only, the area - latency product shows that the FFT implementation is still competitive, despite the fact that the polynomial multiplication achieved is larger, namely for degree-4095 polynomials instead of 2559. However, Karatsuba is much better for the relinearization/key switching, because the FFT fails in terms of flexibility.

## IV. Conclusion

In this paper, we demonstrate that for some cases, Karatsuba algorithm can be a good alternative to FFT/NTT. The study provides 3 architectures for speeding up degree-2559 polynomials multiplication and the relinearization/key switching with coefficients up to 135 bits, with a software/hardware co-design approach. Proposed architectures perform respectively 3, 4 and 5 Karatsuba pre- and post-recursions, plus the sub-polynomials multiplication.
For the first design with 3 recursions, the accelerator can perform a polynomial multiplication in 2.46 ms and a relinearization/key switching in 2.28 ms, which is already competitive compared to FFT/NTT implementation in [1] due to the poor flexibility to perform efficiently the relinearization/key switching. Remaining Karatsuba implementations keep reducing the computational time, at a cost of doubling hardware resources between each additional recursion. For 5 recursions in hardware, the polynomial multiplication is performed in 1.24 ms and the relinearization/key switching in 2.34 ms, which halves computation times compared to the first Karatsuba design. Thus, the paper provides some key information in order to select an approach depending on the application constraint, namely hardware resources available.
Future work will consist on developing a complete demonstrator using the provided Karatsuba implementation.

## Acknowledgment

This study has been partially funded by the french Direction Générale de l'Armement (DGA).

## References

[1] T. Pöppelmann, M. Naehrig, A. Putnam, and A. Macias, "Accelerating homomorphic evaluation on reconfigurable hardware," in *Proc. of Cryptographic Hardware and Embedded Systems – CHES 2015*. Springer, pp. 143–163.
[2] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Proc. of Advances in Cryptology – EUROCRYPT 2010*. Springer, pp. 24–43.
[3] J.-S. Coron, T. Lepoint, and M. Tibouchi, "Scale-invariant fully homomorphic encryption over the integers," in *Proc. of Public-Key Cryptography – PKC 2014*. Springer, pp. 311–328.
[4] J. W. Bos, K. Lauter, J. Loftus, and M. Naehrig, "Improved security for a ring-based fully homomorphic encryption scheme," in *Proc. of Cryptography and Coding: 14th IMA International Conference – IMACC 2013*. Springer, pp. 45–64.
[5] Y. Doröz and B. Sunar, "Flattening ntru for evaluation key free homomorphic encryption," Cryptology ePrint Archive, Report 2016/315, 2016.
[6] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," in *Proc. of the 3rd Innovations in Theoretical Computer Science Conference – ITCS 2012*, pp. 309–325.
[7] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical gapsvp," in *Proc. of Advances in Cryptology – CRYPTO 2012*.
[8] J. Fan and F. Vercauteren, "Somewhat Practical Fully Homomorphic Encryption," Cryptology ePrint Archive, Report 2012/144, 2012.
[9] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Proc. of Advances in Cryptology – CRYPTO 2013*, pp. 75–92.
[10] Z. Brakerski and V. Vaikuntanathan, "Lattice-based fhe as secure as pke," in *Proc. of the 5th Conference on Innovations in Theoretical Computer Science – ITCS 2014*. ACM, pp. 1–12.
[11] A. Khedr, G. Gulak, and V. Vaikuntanathan, "Shield: Scalable homomorphic implementation of encrypted data-classifiers," *Accepted to IEEE Transactions on Computers*, 2015.
[12] J. Pollard, "The Fast Fourier Transform in a Finite Field," in *Mathematics of Computation*, 1971, vol. 25, no. 90, pp. 365–374.
[13] A. Aysu, C. Patterson, and P. Schaumont, "Low-cost and area-efficient FPGA implementations of lattice-based cryptography," in *Proc. of IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2013, pp. 81–86.
[14] N. Göttert, T. Feller, M. Schneider, J. Buchmann, and S. Huss, "On the Design of Hardware Building Blocks for Modern Lattice-Based Encryption Schemes," in *Proc. of Cryptographic Hardware and Embedded Systems — CHES 2012*, ser. LNCS, no. 7428, pp. 512–529.
[15] J. A. Solinas, "Generalized mersenne prime," in *Encyclopedia of Cryptography and Security*. Springer, 2011, pp. 509–510.
[16] S. Sinha Roy, K. Järvinen, F. Vercauteren, V. Dimitrov, and I. Verbauwhede, "Modular hardware architecture for somewhat homomorphic function evaluation," in *Proc. of Cryptographic Hardware and Embedded Systems – CHES 2015*. Springer, pp. 164–184.
[17] D. Pamuła, E. Hrynkiewicz, and A. Tisserand, "Analysis of gf $2^{233}$ multipliers regarding elliptic curve cryptosystem applications." vol. 45, no. 7, pp. 271 – 276, 2012.
[18] T. Lepoint and M. Naehrig, "A Comparison of the Homomorphic Encryption Schemes FV and YASHE," in *Proc. of AFRICACRYPT 2014*, ser. LNCS, vol. 8469, pp. 318–335.
[19] N. P. Smart and F. Vercauteren, "Fully homomorphic simd operations," *Designs, Codes and Cryptography*, vol. 71, no. 1, pp. 57–81, 2014.