High-Speed Fully Homomorphic Encryption over the Integers

Xiaolin Cao, Ciara Moore, Máire O'Neill, Neil Hanley, and Elizabeth O'Sullivan

Centre for Secure Information Technologies, Queen's University Belfast, Northern Ireland

Abstract. A fully homomorphic encryption (FHE) scheme is envisioned as a key cryptographic tool in building a secure and reliable cloud computing environment, as it allows arbitrary evaluation of a ciphertext without revealing the plaintext. However, existing FHE implementations remain impractical due to very high time and resource costs. To the authors' knowledge, this paper presents the first hardware implementation of an encryption primitive for FHE over the integers using FPGA technology. A large-integer multiplier architecture utilising Integer-FFT multiplication is proposed, and a large-integer Barrett modular reduction module is designed incorporating the proposed multiplier. The encryption primitive used in the integer-based FHE scheme is designed employing the proposed multiplier and modular reduction modules. The designs are verified using the Xilinx Virtex-7 FPGA platform. Experimental results show that a speed improvement factor of up to 44 is achievable for the hardware implementation of the FHE encryption scheme when compared to its corresponding software implementation. Moreover, performance analysis shows further speed improvements of the integer-based FHE encryption primitives may still be possible, for example through further optimisations or by targeting an ASIC platform.

1 Introduction

Fully homomorphic encryption (FHE) is a significant breakthrough in cryptographic research in recent years [1]. A FHE scheme can be used to arbitrarily perform computations on a ciphertext without compromising the content of the corresponding plaintext. Thus, a practical FHE scheme will open the door to numerous new security technologies and privacy related applications, such as privacy-preserving search and cloud-based computing.

A working example of FHE was introduced by Gentry in 2009 [2]. Since then, several FHE schemes and corresponding software implementations based on various computationally hard problems have been proposed [1] - [12]. The first software implementation of the lattice-based FHE scheme was reported by Gentry and Halevi (GH) with a public key size ranging from 17 Megabytes (MB) to 2.3 Gigabytes, and a ciphertext homomorphic evaluation time of 6 seconds to 30 minutes [7]. The FHE scheme over the integers was introduced in 2010 by

van Dijk *et al.* [3] and Coron *et al.* [9] then extended this scheme by reducing the public key size, resulting in a bitwise encryption time ranging from 0.05 seconds to 3 minutes. Coron *et al.* [10] further reduced the public key size to no more than 10.1 MB with a longer encryption time, ranging from 0.05 seconds to 7 minutes. A recent FHE software implementation was on an NVIDIA C2050 GPU [13]; it uses the Integer-FFT algorithm [14] for multiplication, the Montgomery algorithm to perform modular reduction in the Integer-FFT execution and Barrett modular reduction [15] to implement the GH FHE scheme [7]. This implementation resulted in a speed improvement of almost 7 compared to the original results [7]. However, results show there is still a long way to go before a practical FHE scheme can be deployed in real-life applications.

To date, there have been few hardware implementations of FHE schemes. Cousins *et al.* [16,17] proposed a hardware implementation on an FPGA platform using the Matlab HDL Coder tool; however they do not report any implementation or simulation results. More recently, an ASIC implementation of a multiplier for the GH FHE scheme is proposed by Doröz *et al* [18]; this implementation shows comparable performance to the original software implementation and uses less area. An FPGA implementation of a multiplier for the GH FHE scheme was proposed by Wang and Huang [19] and is stated to be about twice as fast as the previously mentioned GPU implementation [13]. Further to this, an ASIC design of the full GH FHE scheme, without key generation, is proposed in [20]. Timings show this ASIC implementation is considerably faster than the original implementation in software [7] and also the encryption and recrypt steps are faster than for the GPU platform implementation [13].

The objective of this paper is to accelerate the encryption primitives in integer-based FHE using FPGA technology. This particular FHE algorithm is chosen because of the comparatively simpler theory, smaller key size and comparable performance to the GH scheme. Moreover, the introduction of a batched FHE scheme over the integers promises further efficiency improvements [21]. Multiplication is a key element in these FHE schemes and features in the encryption, decryption and evaluation steps. Large-integer FFT multiplication has also been used in the previously mentioned hardware and GPU implementations of other FHE schemes. In this paper we focus initially on the hardware architecture of a large integer multiplier using the FFT algorithm and how this can speed up the encryption step of an integer-based FHE scheme. Future work will investigate the impact of the hardware multiplier on the other steps within the FHE scheme.

Specifically, we present the first hardware implementation of an encryption primitive required for FHE over the integers. Our contributions are as follows: (i) a novel large-integer hardware multiplier architecture using the Integer-FFT multiplication algorithm is proposed; (ii) a large-integer architecture of Barrett modular reduction using the proposed multiplier as a sub-module is presented along with an analysis of the suitability of four different moduli; (iii) the first hardware architecture for the encryption primitive of FHE over the integers is designed utilising the proposed multiplier and modular reduction; (iv) our

implementations are verified for a Xilinx Virtex-7 FPGA, and the results show our design achieves a significant performance improvement of a factor of 44.72 over equivalent software implementations. An extended version of this work is available on the ePrint Archive [22], where the encryption primitives of two integer-based schemes [9,10] are implemented; there was little difference between the synthesis implementation results of these primitives and thus this paper only presents results for the implementation of the integer-based scheme [9] with the fastest running time.

The rest of the paper is organised as follows. In Section 2, the background information is introduced. In Section 3 the proposed hardware architectures of the FHE encryption primitive is described. Implementation and performance results are given in Section 4. Finally, Section 5 concludes the paper.

2 Related Work

2.1 Encryption Primitive in FHE over the Integers

First proposed by van Dijk *et al.* [3], the integer-based FHE scheme was later improved by Coron *et al.* [9,10] by reducing the public key size. The encryption primitive of the scheme [9], denoted by CMNT here, is implemented in this paper. The encryption step is as follows:

$$C = (M + 2R + 2 \sum_{1 \leq i,j \leq \theta} B_{i,j} \times A_{i,0} \times A_{j,1}) \bmod A_0 \tag{1}$$

where C denotes the ciphertext; $M \in \{0,1\}$ is a 1-bit plaintext; R is a random signed integer in the range $(-2^{\rho}, 2^{\rho})$ and $A_0 \in [0, 2^{\varphi})$ is a part of the public key. $\{B_{i,j}\}$ with $1 \leq i,j \leq \theta$ is a random integer sequence, and each $B_{i,j}$ is a δ-bit integer. $\{A_{i,0}\}$ and $\{A_{i,1}\}$ with $1 \leq i \leq \theta$ are two public key sequences, and each entry is a φ-bit integer. The parameter bit-lengths of four test groups [9], used in the performance comparison in Section 4, are listed in Table 1. For further information on the parameter selection and the parameter security levels, see [9,10].

Table 1: Four Parameter Groups for CMNT FHE Scheme

Group	δ	$\rho = 4\delta$	$\varphi \times 10^{-6}$	θ
Toy	42	168	0.16	12
Small	52	208	0.86	23
Medium	62	248	4.20	44
Large	72	288	19.0	88

To implement (1), the first challenge is the very large multiplication. The multiplication algorithm typically used for very large bit-length operands is the

Integer-FFT [14,23,24]. It conquers large bit-length multiplication by dividing it into smaller bit-length multiplication and then accumulating. For example, the widely used open-source GMP library uses the Schönhage-Strassen Integer-FFT algorithm [14] for multiplication when the operand bit-length is greater than 2^{15} bits [25]. There exist many Integer-FFT variants which aim to improve the speed of small bit-length multiplication, as it is the bottleneck of the Integer-FFT algorithm. However, the use of the embedded multipliers on a Xilinx Virtex-7 FPGA can address this issue, as they are specifically optimised for high-speed performance of up to 750 MHz [26]. Thus, the basic Integer-FFT algorithm [24] combined with these embedded multipliers is used in our work.

The large modular reduction is also a considerable challenge. Generally, the modular reduction algorithms used in traditional long bit-length cryptographic implementations are Montgomery [27] and Barrett reduction [15]. However, due to heavy pre-computation and post-processing costs, Montgomery reduction is only suitable in scenarios where successive modular operations with the same operands are required, such as exponentiation for example. In contrast, Barrett reduction only requires a one-time pre-computation, and therefore is used in our implementation.

The objective of this work is to accelerate the speed of the encryption step outlined in (1) rather than deal with storage bottlenecks. Therefore, it is assumed that there is sufficient off-chip memory available for the designed FPGA accelerator to store intermediate variables and final results. This is a reasonable assumption as the accelerator can be viewed as a powerful coprocessor device, sharing memory with a main workstation over a high speed PCI bus. However, it is acknowledged that with off-chip memory I/O can become a bottleneck and the latency of the bus becomes an issue. Investigations into such issues will be the subject of future work.

2.2 The Integer-FFT Multiplication Algorithm

Integer-FFT multiplication treats each multiplication operand as a sequence of smaller, computationally efficient numbers instead of a single large integer. The input parameters for Integer-FFT multiplication are:

- p, an m-bit number, the modulus in Integer-FFT modular reduction
- q_i, the prime factors of p
- k, the FFT point number
- ω, the twiddle factor of the FFT
- b, the base unit bit-length when transforming the input operand into a b-bit digit sequence

To ensure the Integer-FFT algorithm works correctly, the FFT point number k must divide $q_i - 1$ for every prime factor q_i of p (if p is a prime, q_i equals p). The twiddle factor ω is a primitive k^{th} root of unity, that is $\omega^k \equiv 1 \,(\mathrm{mod\,p})$ and $\omega^{k/q_i} - 1 \not\equiv 0 \,(\mathrm{mod\,p})$ for any prime divisor q_i of p [14], and all operations used in the FFT should be modular with respect to the modulus p. The requirement for a suitable base unit bit-length is $k(2^b - 1)^2/2 < p$ [24].

Table 2: The Four Integer-FFT Moduli

Group	p	m	k	ω	b
Special Modulus [28]	$2^{32}+1$	33	64	2	8
Special Modulus [28]	$2^{64}+1$	65	128	2	24
Solinas Modulus [24, 29]	$2^{64}-2^{32}+1$	64	128	7	28
General Modulus [24]	$2^{32}-2^{20}+1$	32	64	17	12

As the selection of a reasonable modulus, p, heavily influences the modular multiplication performance, four different moduli, as defined in Table 2, are implemented and compared in this work. Further details on the moduli are given in Section 3.3. The Integer-FFT algorithm [14] is outlined in Algorithm 1.

Algorithm 1: Integer-FFT Multiplication [24]

Input: x, y, b, z
Output: $c = x \times y$

1 Compute the FFT of the digits of x and y, with respect to the base b, where each digit is treated as a FFT sample;
2 Multiply the FFT results component by component: $z_i = FFT(x_i) \times FFT(y_i)$;
3 Compute the inverse FFT (IFFT): $c_i = IFFT(z_i)$;
4 Resolve the carry chain: when $c_i \geq b$, set $c_{i+1} = c_i \,\mathbf{div}\, b$ and set $c_i = c_i \,\mathbf{mod}\, b$;
5 **return** c

2.3 The Barrett Modular Reduction

Two Barrett modular reduction hardware architectures have been designed for this work. The first is for small integer reduction used in the Integer-FFT algorithm, and the second is for the proposed large integer Barrett reduction design. Both adopt the Barrett reduction algorithm introduced in [30], outlined in Algorithm 2.

The essence of Barrett reduction is that the intermediate parameter $\hat{\sigma}$ given in Algorithm 2 is used to estimate x/p, where $\lfloor . \rfloor$ is the floor operation. Then $x - \hat{\sigma}p$ is used to approximate $x (\mathrm{mod}\, \mathrm{p})$. The advantage of this algorithm is that it has been proven that if $\beta < -2$ and $\alpha > m$, at most one subtraction is required in the final reduction [30].

3 The Proposed FHE Encryption Architecture

3.1 Multiplier Architecture Overview

The multiplier architecture consists of a shared RAM, a finite state machine (FSM) controller and an Integer-FFT unit. The shared RAMs are assumed to

Algorithm 2: Barrett Reduction Algorithm

Input: x ($2m$-bit), p (m-bit) and a pre-computed constant $p_1 = \lfloor 2^{m+\alpha}/p \rfloor$
Output: $y = x \pmod p$

1 $\hat{\sigma} = \left\lfloor \frac{\lfloor x/2^{m+\beta} \rfloor \times \lfloor 2^{m+\alpha}/p \rfloor}{2^{\alpha-\beta}} \right\rfloor$
2 $p_2 = \hat{\sigma} \times p$;
3 $y_1 = x - p_2$ and $y_2 = y_1 - p$;
4 If $y_2 < 0$, $y = y_1$, otherwise $y = y_2$;
5 **return** y

be off-chip and are used to store the input operands, the intermediate results and final results. The FSM controller is responsible for distributing the signals to schedule the algorithm. The proposed FSM scheduling mechanism can be viewed as a combination of school-book [25] and Integer-FFT [14] multiplication. The core element of the design is an Integer-FFT module that executes a block multiplication to calculate partial products, while the FSM controller schedules an iterative school-book multiplication to accumulate the block products. The proposed architecture, depicted in Fig 1, is fully pipelined and the RAM read, RAM write and Integer-FFT operations are executed in parallel.

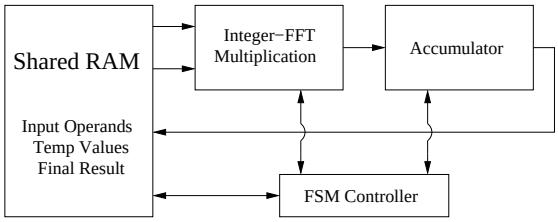


Fig. 1: Overview of the proposed hardware multiplier

3.2 The FFT/IFFT Module and Butterfly Unit

Various FFT algorithms and architectures can be used to implement the FFT algorithm for different optimisation goals [23, 24]. In this work, a radix-2 fully parallel architecture is adopted for FFT and IFFT in order to obtain the highest throughput. There are $\log_2 k$ butterfly stages for a k-point FFT, and each butterfly stage is composed of $k/2$ parallel butterfly units.

The IFFT needs to multiply $k^{-1} (\mathrm{mod}\, p)$, which is not required in the FFT. If an identical architecture is used to implement both the IFFT and FFT, a point-wise module multiplication stage is additionally required for the IFFT and the cycle latency is increased. This problem can be solved by pre-computing $\hat{\omega}^{-1} = (k\omega)^{-1} (\mathrm{mod}\, p)$ to incorporate $k^{-1} (\mathrm{mod}\, p)$ into the IFFT twiddle factors,

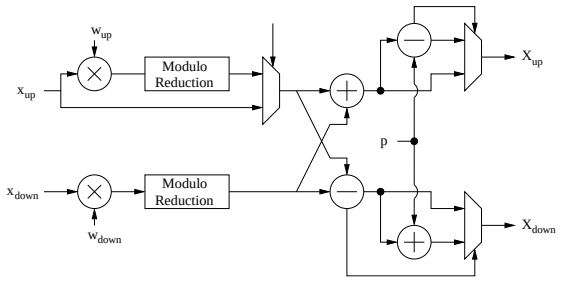


Fig. 2: The proposed FFT/IFFT butterfly unit

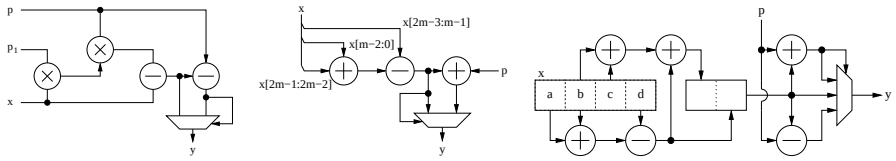
then $\hat{\omega}^{-1}$ is used in the final IFFT butterfly stage, while the previous stages use ω^{-1}. In order to meet the butterfly requirement of both FFT and IFFT, a unified butterfly unit is proposed in Fig. 2. The multiplication at the bottom left-hand side in Fig. 2, $x_{down} \times \omega_{down}$, is the same for all FFT/IFFT butterfly stages, as is the operation of X_{up}/X_{down} on the right-hand side of Fig. 2. However, the operation of $x_{up} \times \omega_{up}$ on the upper left-hand side of Fig. 2 is only required in the final stage of the IFFT.

In the design presented here, if the special modulus $p = 2^{m-1} + 1$ is used, each m-bit multiplier in a butterfly is implemented using a bit-shift operation, as the k^{th} primitive root of unit, ω, equals 2, as stated in Table 2. Otherwise, each butterfly multiplier is designed using a multi-stage pipelined multiplier and implemented using FPGA embedded multipliers using Xilinx Core Generator [26] tools. This prevents the multipliers becoming the bottleneck in our design.

3.3 The Modular Reduction Module

The small integer modular reduction is very simple, only requiring an addition/subtraction operation; it is illustrated in the right half of Fig. 2. Therefore, this subsection introduces the modular reduction unit used after the butterfly and point-wise multiplication. Three modular reduction methods are designed and tested in our work: Barrett reduction with any modulus (in this case $2^{32} - 2^{20} + 1$); the simplest reduction method with the modulus in the special form $2^{m-1} + 1$; and modular reduction using the suitable Solinas modulus, $2^{64} - 2^{32} + 1$ [29].

The Barrett reduction architecture is shown in Fig. 3(a) and it requires two multipliers and two subtractions. Following Algorithm 2, in our design we set $\beta = -4$ and $\alpha = m + 4$; thus the pre-computed constant is $p_1 = 2^{2m+4}/p$. The second design with special form modulus [14] is shown in Fig. 3(b) and the reduction $y = x (\mathrm{mod}\, p)$ is easily obtained using the logic in Fig.3(b) as follows: let $y_1 = x\,[m-2\colon 0] + x\,[2m-1\colon 2m-2] - x\,[2m-3\colon m-1]$ and $y_2 = y_1 + p$; if $y_1 < 0$, $y = y_2$; else $y = y_1$ [31]. As no multiplication is required, this circuit consumes less hardware resources than traditional Barrett reduction and offers faster performance.



(a) (b) (c)

Fig. 3: Proposed modular reductions used in FFT butterfly and point-wise multiplication: (a) Barrett reduction suitable for all moduli; (b) the simplest reduction for special form moduli, $p = 2^{m-1}+1$; (c) a simpler reduction for Solinas moduli

In Fig. 3(c) the Solinas modulus $p = 2^{64} - 2^{32} + 1$ is used and the 128-bit multiplication can be expressed as $x = 2^{96}a + 2^{64}b + 2^{32}c + d$, where a, b, c and d are 32-bit numbers. As $2^{96} \equiv -1 (\mathrm{mod}\, p)$ and $2^{64} \equiv 2^{32} - 1 (\mathrm{mod}\, p)$, the Solinas modular reduction can be quickly computed as $x \equiv 2^{32}(b+c) - a - b + d (\mathrm{mod}\, p)$. Since the result, $2^{32}(b+c) - a - b + d$, is within the range $(p, 2p)$, only an addition, a subtraction and a $3 \rightarrow 1$ multiplexer are needed for the reduction. This is still much simpler than Barrett reduction as no multiplication is required. However, given the modulus and twiddle factor restrictions described in Section 2, not every Solinas modulus is suitable [24].

The FHE encryption architecture is tightly coupled with the large Barrett modular reduction using the FSM controller since only one instance of the proposed multiplier is implemented; the FSM controller occupies a very small percentage of area compared to the multiplier. The encryption primitive in (1) is implemented by firstly executing the pipelined multiplier and accumulation modules in parallel and secondly performing the large Barrett reduction.

4 Implementation, Performance and Comparison

The proposed architectures are designed and implemented using Xilinx FPGA technology. The synthesis tool used is Xilinx ISE Design Suite 14.1 and Modelsim 6.5a is used as the functional and post-synthesis timing simulation tool. The optimisation objective of the synthesis tool is set to speed. The target device is the Virtex-7 XC7VX980T-2FFG1926. The test vectors are generated as random numbers using C++ according to the parameters given in Table 1.

The proposed multiplier architecture is implemented as a fully pipelined and parallel circuit. At the outer interface, three RAM read buses and one RAM write bus are implemented, so large multiplication operands can be read into the multiplier and final block product accumulation can be executed simultaneously. At the inner layer, the Integer-FFT multiplier is also pipelined. The basic computation bit-length in the Integer-FFT is determined by modulus bit-length rather than base unit bit-length, so increasing base unit bit-length will not reduce speed performance. However, base unit bit-length is related to the multiplication result, so the larger the base unit bit-length, the lower the latency.

In our implementations the ratio of multiplication operand block bit-length $kb/2$ and data bus bit-width d, $kb/2d$, is equal to 8. The data bus bit-width is then equal to 32, 192, 224 and 48 for the moduli respectively. The clock cycle count therefore equals 8 for each $\frac{kb}{2}$-bit input operand, and for each output block product except the first block product. The pipeline stage number in each FFT butterfly stage is designed to also equal 8.

We have implemented the FHE encryption primitive with the moduli listed in Table 2 and the synthesis results are shown in Table 3. It should be noted that, in general, post place and route of the design will give slightly worse results. For each design, the FFT point number k is fixed and is unrelated to multiplication operand bit-length.

Table 3: Synthesis Results of FHE Encryption Primitive

Integer-FFT modulus	Frequency (MHz)	# DSPs	# Slice registers	# Slice LUTs
Special: CMNT $2^{32}+1$	292.410	256	191176	237031
Special: CMNT $2^{64}+1$	179.346	2048	956974	1215166
Solinas: CMNT $2^{64}-2^{32}+1$	166.450	18496	1123001	954955
General: CMNT $2^{32}-2^{20}+1$	254.054	6292	213788	171450

For the small integer multiplications used in the FFT butterfly and point-wise multiplication, Xilinx Core Generator is employed to generate a 4-stage pipelined multiplier using Virtex-7 FPGA embedded multipliers. From Table 3 the special modulus design with modulus $2^{32}+1$ requires the least hardware resources, as no multiplication is needed in the FFT butterfly unit. Also, this design which uses the modulus with the smallest bit-length has the highest frequency, as the multipliers needed for point-wise multiplication in the Integer-FFT algorithm contain the critical path and are implemented as 4-stage pipelined multipliers. However this does not mean that the special modulus design will offer the best performance, as the other moduli allow larger base unit bit-length, which implies that in one clock cycle the Solinas modulus multiplier can produce the longest product. We also find that only the implementation with the smallest modulus, $2^{32}+1$, is within the hardware resource budget of the targeted XC7VX980T FPGA device. Moreover the implementation with the second special modulus, $2^{64}+1$, can fit on the largest available XC7V2000T FPGA device. Thus an ASIC platform would be required for the implementations with the other two moduli; however this would mean the design would no longer be able to take advantage of the embedded multipliers available on Xilinx Virtex-7 FPGAs.

The running time of the proposed hardware implementation of the encryption primitive using the parameter groups from Table 1 is compared with the corresponding previously reported software results in Table 4. The running time is obtained by averaging the latency of the simulated test vectors and multiplying by the clock frequency. It can be seen that the special modulus multiplier

has a higher frequency but still requires more execution time than the Solinas modulus multiplier. This is because in the multiplication circuit, the throughput is mainly determined by the product of data bus bit-width, base unit bit-length and circuit frequency, rather than just the frequency. Comparing moduli, it is also clear to see that although the special moduli can use a simple twiddle factor of 2, its base unit bit-length is much smaller than for the other modulus types when almost the same bit-length modulus is employed. Therefore the Solinas modulus is the best choice of moduli, enabling a comparatively simpler modular reduction and a larger base unit bit-length.

Table 4: Average Running Time of Proposed FHE Encryption Design

Integer FFT Modulus	Toy	Small	Medium	Large
CMNT $2^{32}+1$	0.003 s	0.050 s	0.872 s	15.735 s
CMNT $2^{64}+1$	0.000854 s	0.0139 s	0.239 s	4.284 s
CMNT $2^{64}-2^{32}+1$	0.000815 s	0.0130 s	0.221 s	3.958 s
CMNT $2^{32}-2^{20}+1$	0.003 s	0.057s	1.003 s	18.110 s
CMNT on Intel Core2 Duo E8500 [9]	0.05 s	0.79 s	10 s	2 min 57 s

Table 4 also shows that a hardware design of the encryption step in integer-based FHE with a Solinas modulus is 61.35 and 44.72 times faster than the corresponding software implementation of CMNT for the toy and large parameter groups respectively. Moreover, for the implementations with special moduli $2^{32}+1$ and $2^{64}+1$, which both fit on to FPGA devices, speed up factors of 11.25 and 41.32 are achieved for the large parameter group respectively. It must be noted that we only give experimental results using small FFT parameters (i.e., $k \leq 128$ and $m \leq 65$). As the product of data bus bit-width and frequency determines multiplier performance, we believe that there is much potential for speed improvement of the integer-based FHE encryption primitives if larger FFT parameters are used. Ongoing work focuses on the use of a Solinas prime modulus for a lower cost design, that can achieve a comparable speed to this implementation.

5 Conclusion

In this paper, the first hardware implementations of the encryption primitive employed in the integer-based FHE scheme by Coron *et al.* [9] are presented. For this purpose, an Integer-FFT based hardware multiplier module and a Barrett modular reduction module are proposed. These hardware architectures are designed and verified on a Xilinx Virtex-7 device. When the encryption primitive is implemented with a particular Integer-FFT modulus, such as the special modulus $2^{32}+1$, the synthesis results show that a speed improvement factor of up to 11.25 is possible compared to the corresponding software implementation for

the large scale test data used in FHE over the integers; moreover this design fits on a Virtex-7 FPGA device. This speed improvement factor could be increased to at least 44 if another Integer-FFT modulus such as the Solinas modulus is used; however an ASIC device would need to be targeted, which would provide further inherent improvements in speed. The modulus size is limited in terms of practical or implementable FPGA design due to the excessive hardware cost. As our implementations only use at most 128-point FFT and small base unit bit-lengths of at most 28 for the proposed hardware multiplier, there is still potential to further improve the encryption speed in FHE over the integers by increasing the FFT point and targeting an ASIC platform.

References

1. Gentry, C.: A fully homomorphic encryption scheme. Ph.D. thesis, Stanford University, 2009. [Online]. Available: http://crypto.stanford.edu/craig.
2. Gentry, C.: Fully homomorphic encryption using ideal lattices. Proc. the 41st annual ACM symposium on Theory of Computing, pp. 169-178. (2009)
3. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. EUROCRYPT 2010, Springer, LNCS, vol. 6110, pp. 24-43. (2010)
4. Smart, N. P. and Vercauteren, F.: Fully homomorphic encryption with relatively small key and ciphertext sizes. PKC 2010, Springer, LNCS, vol. 6056, pp. 420-443. (2010)
5. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. Advances in Cryptology - CRYPTO 2011, Springer, LNCS, vol. 6841, pp. 505-524. (2011)
6. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption for ring-LWE and security for key dependent messages. CRYPTO 2011, Springer, LNCS, vol. 6841, pp. 505-524. (2011)
7. Gentry, C., Halevi, S.: Implementing Gentry's fully homomorphic encryption scheme. EUROCRYPT 2011, Springer, LNCS, vol. 6632, pp. 129-148. (2011)
8. Brakerski, Z., Gentry, C, Vaikuntanathan, V.: Fully homomorphic encryption without bootstrapping. Cryptology ePrint Archive, Report 2011/277 (2011)
9. Coron, J. S., Mandal, A., Naccache, D., Tibouchi, M.: Fully homomorphic encryption over the integers with shorter public keys, CRYPTO 2011, Springer, LNCS, vol. 6841, pp. 487504. (2011)
10. Coron, J. S., Naccache, D., Tibouchi, M.: Public key compression and modulus switching for fully homomorphic encryption over the integers. EUROCRYPT 2012, Springer, LNCS, pp. 446-464. (2012)
11. Lauter, K., Naehrig, M., Vaikuntanathan, V.: Can homomorphic encryption be practical? Cryptology ePrint Archive, Report 2011/405 (2011)
12. Gentry, C., Halevi, S., Smart, N. P.: Homomorphic evaluation of the AES circuit. Cryptology ePrint Archive, Report 2012/099 (2012)
13. Wang, W., Hu, Y., Chen, L. Huang, X., Sunar, B.: Accelerating fully homomorphic encryption using GPU. High Performance Extreme Computing Conference 2012, IEEE, pp. 1-5. (2012)
14. Schönhage, A., Strassen, V.: Schnelle Multiplikation grosser Zahlen. Computing, Springer, vol. 7, no. 3, pp. 281-292. (1971)

15. Barrett P.: Implementing the Rivest, Shamir and Adleman public key encryption algorithm on a standard digital signal processor. CRYPTO 1986, Springer, pp. 311-323. (1986)
16. Cousins, D. B., Rohloff, K., Peikert, C., Schantz, R.: SIPHER: Scalable implementation of primitives for homomorphic encryption - FPGA implementation using Simulink. IEEE High Performance Extreme Computing Conference (2011)
17. Cousins, D. B., Rohloff, K., Peikert, C., Schantz, R.: SIPHER: An update on SIPHER (Scalable Implementation of Primitives for Homomorphic EncRyption) - FPGA implementation using Simulink. IEEE Conference on High Performance Extreme Computing, pp. 1-5. (2012)
18. Doröz, Y., Öztürk, E., Sunar, B.: Evaluating the Hardware Performance of a Million-Bit Multiplier. Digital System Design, pp. 955-962. (2013)
19. Wang, W., Huang, X.: FPGA implementation of a large-number multiplier for fully homomorphic encryption. International Symposium on Circuits and Systems, pp. 2589-2592. (2013)
20. Doröz, Y., Öztürk, E., Sunar, B.: Accelerating Fully Homomorphic Encryption in Hardware. Under review [Online]. Available: http://ecewp.ece.wpi.edu/wordpress/vernam/files/2013/09/Accelerating-Fully-Homomorphic-Encryption-in-Hardware.pdf.
21. Cheon, J. H, Coron, J. S., Kim, J., Lee, M. S., Lepoint, T., Tibouchi, M., Yun, A.: Batch fully homomorphic encryption over the integers. Advances in Cryptology - EUROCRYPT 2013, Springer, LNCS, vol. 7881, pp. 315-335. (2013)
22. Cao, X., Moore, C., O'Neill, M., O'Sullivan, E., Hanley, N. Accelerating fully homomorphic encryption over the integers with super-size hardware multiplier and modular reduction. Cryptology ePrint Archive, Report 2013/616 (2013)
23. Craven, S., Patterson, C., Athanas, P.: Super-sized multiplies: how do FPGAs fare in extended digit multipliers? 7th International Conference on Military and Aerospace Programmable Logic Devices (2004)
24. Emmart, N., Weems, C.: High precision integer multiplication with a GPU using Strassen's algorithm with multiple FFT sizes. Parallel Processing Letters, vol. 21, no. 3, pp. 359. (2011)
25. GMP, The GNU Multiple Precision Arithmetic Library [Online]. Available: http://gmplib.org/manual/Multiplication-Algorithms.html, Multiplication Algorithms.
26. Xilinx Product Specification: LogiCORE IP Multiplier v11.2. [Online]. Available: http://www.xilinx.com/support/documentation/ip-documentation/mult-gen-ds255.pdf.
27. Montgomery, P.: Modular multiplication without trial division. Mathematics of computation, vol. 44, no. 170, pp. 519-521. (1985)
28. Kalach, K., David, J. P.: Hardware implementation of large number multiplication by FFT with modular arithmetic. 3rd International IEEE-NEWCAS Conference, pp.267-270. (2005)
29. Solinas, J. A.: Generalized Mersenne Numbers. Issue 39 of Research report, University of Waterloo. Faculty of Mathematics (1999)
30. Dhem, J. F.: Design of an efficient public-key cryptographic library for RISC-based smart cards. PhD thesis, Université catholique de Louvain,1998. [Online]. Available: http://users.belgacom.net/dhem/these/
31. Zimmermann, R.: Efficient VLSI implementation of modulo $(2^n \pm 1)$ addition and multiplication. IEEE Symposium on Computer Arithmetic, pp.158-167. (1999)