Efficient Fast Additive Homomorphic Encryption Cryptoprocessor for Privacy-preserving Federated Learning Aggregation

Wenye Liu, Nazim Altar Koca, and Chip-Hong Chang
Email: wenye.liu@ieee.org, koca0001@e.ntu.edu.sg, echchang@ntu.edu.sg

***Abstract*—Privacy leakage is a critical concern of collaboratively training a large-scale deep learning model from multiple clients. To protect the local data, homomorphic encryption (e.g., Paillier) could be utilized for data aggregation on the central server. Nevertheless, even with CPU-optimized libraries or FPGA-based accelerators, the computing power and throughput limitations remain a stumbling block for practical deployment of Paillier scheme. In this paper, we present an efficient and high-throughput cryptoprocessor based on a recently introduced Fast Additive Homomorphic Encryption (FAHE) algorithm. For encryption, we incorporate the asymmetric decomposition, time multiplexing resource reuse and hard-macro based wide-bus logic operations to efficiently map the large (>40 kbits) integer multiplications for low latency FPGA implementation. For decryption, we propose a table lookup method for rapid modular reduction by leveraging the relative short modulus size of FAHE. The single large precomputed lookup table is carefully partitioned into multiple subtables and deployed in dual-port RAMs to enable resource-efficient parallel computation. The FAHE cryptoprocessor is implemented on a Xilinx ZCU102 FPGA board for performance evaluation and comparison. The results show that the throughput of our design is $354\times$ to $404\times$ higher than the state-of-the-art Paillier accelerators. Compared to the FAHE software implementation, the latency of our proposed design is $14.95\times$ and $11.42\times$ lower for encryption and decryption, respectively.**

***Index Terms*—additive homomorphic encryption, federated learning, FPGA hardware acceleration**

I. Introduction

Large-scale deep learning models show promising performance on practical applications in computer vision and natural language processing [1]. The voracious appetite for data to train large model puts privacy at risk, especially for sensitive data like clinical and financial records [2]. Federated learning (FL) [3] emerges as a new paradigm to collaboratively train a large model without collecting raw data from the clients. Even then, the exchanged model data such as gradients still carries extensive training data information which can lead to privacy breaches under malicious attacks [4]. To prevent leakage of private data, many techniques including differential privacy [5], multi-party computation and homomorphic encryption (HE) [6] have been proposed. Among which, HE is touted to be the most ideal privacy-preserving solution for distributed learning.

HE is a special cryptosystem that allows computations to be performed on the encrypted data. HE may not be practical for adoption in cross-device FL that involves a very large number of portable and IoT devices with tight computing budget and unreliable connections. Cross-silo FL, on the other hand, involves typically a handful to few tens of different organizations (e.g., hospitals or banks) or geo-distributed datacenters. These clients have more computing power and larger bandwidth. Thus, HE is a perfect candidate to provide the much required strong security guarantee for cross-silo FL [7]. Even though fully homomorphic encryption (FHE) [8] is not required in model averaging, partially homomorphic encryption still entails enormous computational resources. Paillier cryptosystem [9] is a representative additive HE for FL. However, it increases the overall training time tremendously [7]. Recent studies [10] show that the cryptographic processing time can be two orders of magnitude higher than gradient computation. To make things worse, the overhead of cryptographic computations is proportional to the size of the model's parameters. The bottleneck operation of Paillier is modular exponentiation with large modulus for encryption and decryption. To improve the efficiency and throughput of Paillier HE, data batching [10] and hardware accelerated approaches were proposed. Parallelism across data items, interleaving and hardware-aware Montgomery modular multiplication [11] are utilized for FPGA implementations [12] [13] [14] [15] of Paillier. Nevertheless, only the implementation of [12] has achieved the 3072-bit key size requirement for a 128-bit security strength [16]. The large key size makes the modular arithmetic extremely complicated and expensive for efficient hardware implementation, particularly on FPGA. Recently, a fast additive homomorphic encryption (FAHE) scheme based on the approximate common divisor problem was proposed [17]. It provides considerable improvement in throughput compared to Paillier. To the best of our knowledge, no hardware-accelerated FAHE has yet been explored or reported.

In this paper, we present the first hardware implementation of FAHE on the FPGA platform. We adopt a scalable architecture in encryption by decomposing the large asymmetric integer ($>$ 40 kbits) multiplication into smaller low complexity multipliers, multiplexing the merging adders in different cycles for resource reuse and exploiting DSP blocks for lucrative implementation of complex high-width logic operations. For the decryption, we partition the operand into multiple chunks for parallel computation with pre-computed data [18]. A large lookup table is split into subtables to optimize the RAM configuration for higher throughput and memory utilization rate. The advantages of short key size in FAHE has been fully utilized in the pipeline dataflow. We have implemented the proposed FAHE cryptoprocessors on the Xilinx Ultrascale+ ZCU102 FPGA board to support up to 512 homomorphic additions with 128-bit security

979-8-3503-4859-0/DATE24/© 2024 EDAA

strength. Compared to the state-of-the-art Paillier accelerator with the same security level, our design achieves at least $404\times$ and $354\times$ speedup for encryption and decryption, respectively. The latency of our proposed cryptoprocessors is $14.95\times$ and $11.42\times$ less than the most efficient CPU implementation of FAHE.

II. Preliminaries

FL can be adopted in cross-device or cross-silo domains. Cross-silo FL involves a small number of companies and organizations like hospitals and financial institutions that have incentive to collaboratively train a shared model. These clients are not constrained by power and bandwidth budgets but have very stringent confidentiality or legal constraints [19]. HE, especially the additive scheme, is a popular approach to protect the privacy of local data in cross-silo setting [10].

A. Homomorphic Encryption

Fully Homomorphic Encryption (FHE) is touted as a cryptography's holy grail that allows arbitrary operations to be computed on encrypted data without decrypting them [8]. Current FHE schemes and the latencies of their implementations [20] are still far from being efficient due to the high computational complexity and large size of parameters. Fortunately, application like cross-silo FL requires only additive HE. Paillier cryptosystem [9] is an additive HE that has been widely used in the FL framework [21]. The key size is set to 3072 bits to achieve a 128-bit security level equivalent of RSA-3072. The encryption key $(n,\ g)$ and the decryption key $(\theta,\ \mu)$ can be generated by randomly choosing two large (1536 bits) prime numbers, P and Q, where $n = PQ$, $g = n + 1$, $\theta = (P-1)(Q-1)$ and $\mu = ((P-1)(Q-1))^{-1}\ mod\ n$. The message $m < n$ can be encrypted as follows:

$$c = g^m \cdot r^n \ \ mod\ \ n^2 \tag{1}$$

where c is the ciphertext, $r < n$ is a random number for each encryption.

The ciphertext can be decrypted as follows:

$$m = L(c^\theta \ \ mod\ \ n^2) \cdot \mu \ \ mod\ \ n \tag{2}$$

where $L(x) = (x-1)/n$. It is worth noting that increasing the length of the key size for n will lead to a quadratic increase in the complexity of exponentiation modulo n^2 operation, and a significant throughput reduction.

Fast Additive Partially Homomorphic Encryption (FAHE) [17] is a lightweight and secure alternative to Paillier. In addition, FAHE scheme designed based on approximate common divisor problem is a post-quantum secure cryptographic primitive. The symmetric key scheme agrees with the FL paradigm where the encryption and decryption are to be performed on the same client machine. With an appropriate choice of parameters, the number of additions allowed by FAHE can be tailored to the cross-silo FL application requirement. Two algorithms, FAHE1 and FAHE2, were proposed in [17]. The main difference is FAHE2 provides shorter ciphertext size with less computing resources but requires stronger security assumption. Since FAHE1 is more general and has stronger security, we choose FAHE1 as the target for acceleration. Three

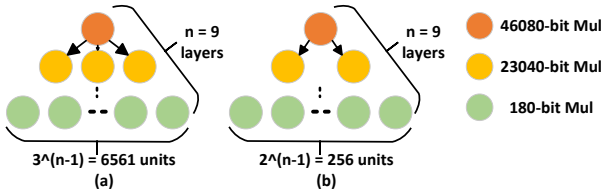

Fig. 1. Multiplier decomposition a) Karatsuba b) proposed simplified decomposition with asymmetric inputs.

parameters need to be set for the key generation of FAHE. The security parameter λ is set to 128 to obtain the same level of security as RSA-3072. The parameter α is determined by the maximum number of homomorphic additions. It is set to 9 to support up to 512 (2^α) additions. The maximum message size m_{max} is set to 32 for a 32-bit integer message. The parameters (ρ, η, γ) are computed by $\rho = \lambda$, $\eta = \rho + 2\alpha + m_{max}$ and $\gamma = \frac{\rho}{log_2(\rho)} \cdot (\eta - \rho)^2$. Finally, a prime number p (i.e., the symmetry key) of size η is selected.

The encryption requires a random number $q \in [0, 2^{\gamma-\eta}]$ (45714 bits), and a noise term of ρ bits. Then the ciphertext c is calculated as:

$$c = (q \cdot p) + (m << (\rho + \alpha)) + noise \tag{3}$$

where m is the message to be encrypted.

The decryption operation of a ciphertext c is given as:

$$m = (c \ \ mod\ \ p) >> (\rho + \alpha) \tag{4}$$

B. Accelerations of Additive HE

The encryption and decryption of gradients can consume more than 80% of the total iteration time in cross-silo FL using Paillier HE [10]. Thus, it is imperative to increase the throughput of HE in FL. To tackle this issue, software library like python-paillier [22] is developed with $8\times$ throughput improvement compared to vanilla python. Intel also provides the Paillier Cryptosystem Library [23] based on AVX512 and IFMA instructions. This library can boost the performance by $4\times$ compared to GMP. However, these speedups can only be obtained on the new generation of Ice Lake-SP or Sapphire Rapids-SP Xeon CPUs.

Other than CPU-optimized libraries, hardware accelerators of Paillier were also proposed. The very first FPGA accelerator of Paillier was proposed in [14], where the modular exponentiation is broken into a series of modular multiplications by a square-and-multiply algorithm for implementation with high-radix Montgomery modular multiplier. The SoC-based accelerator [13] uses a multicore architecture to accelerate the large integer modular arithmetic. Recognizing the difficulty of parallelizing iterative modular multiplications for modular exponentiation in Paillier encryption, Yang et al. [12] optimize each encryption core, making it compact and resource efficient to maximize the number of encryption cores that can be fitted onto a single large FPGA instance. This allows multiple gradient updates to be encrypted in parallel during FL. PipeFL [15] utilizes a hardware-aware Montogomery algorithm for data parallelism and pipelining on FPGA. The design that has the

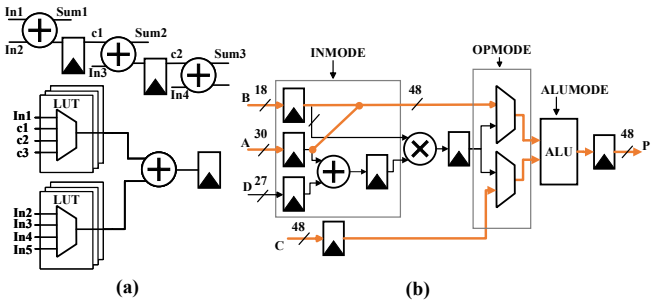

Fig. 2. a) Time multiplexing for adder reuse b) Configuration of the DSP block for wide-bus logic operations.

highest throughput among all the reported Paillier accelerators is presented in [12]. However, its throughput dropped from 5200 OP/s to 1000 OP/s and <500 OP/s when the key size was increased from 1024 bits to 2048 bits and 3072 bits, respectively. Other designs like [13] and [14] are implemented for only up to 2048-bit key size, PipeFL [15] is evaluated for a maximum key size of 1024 bits. Paillier key length of less than 3072 bits cannot meet the 128-bit security strength [16] requirement.

III. Proposed Architecture for FAHE

During FL, the encryption of local updates and the decryption of aggregated updates are performed in different phases. Therefore, the encryption and decryption processors can be designed independently with shared resources on the same FPGA device. The two most compute-intensive operations in FAHE are the large integer multiplication and modular reduction as shown in (3) and (4). Although large integer multiplication is a known performance bottleneck in cryptographic processors, hardware acceleration of extremely large integer arithmetic operations of >40 kbits like those entailed in FAHE is rare. Current studies like [24] and Impress [25] only evaluate up to a maximum bit length of 16384. For decryption, the arithmetic units for a much shorter modulus p (178 bits) of FAHE can be simplified substantially more than those of modulus n^2 (6144 bits) of Paillier. This is exploited in our design of modular reduction unit by using a parallel pipelined table-look-up approach. To maximize the memory efficiency for the implementation of split subtables using FPGA resources, the storage is carefully partitioned to map on dual-port RAMs.

A. FAHE Encryption

We adopt a 46080×180-bit multiplier design for the random number q (45714 bits) and the prime value p (178 bits) used in the FAHE encryption. Direct synthesis of this large asymmetric integer multiplication leads to infeasibly high FPGA resource utilization. An efficient mapping of the architecture to FPGA is required to achieve a very high throughput performance. To this end, three design techniques, namely asymmetric decomposition, time-multiplexing on chain adders and DSP-based wide-bus operation, are proposed.

Karatsuba is a well-known fast multiplication algorithm which uses three half-size multiplications and two additions to form a full-size multiplication. In this design, we further

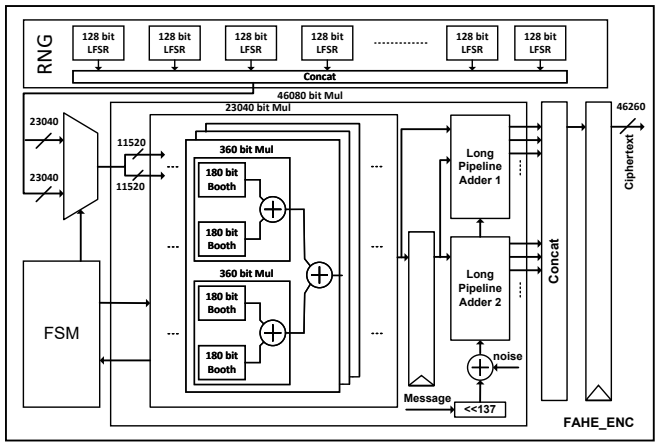

Fig. 3. Architecture of the FAHE encryption processor.

decompose the large asymmetric multiplier using the disproportionately large difference in the size of these two operands. This leads to the use of only two half-size multipliers and one adder for the same multiplication, which results in a significant saving on the logic and routing costs. The comparison of these different decompositions is illustrated in Fig. 1. Karatsuba requires 6561 180-bit multipliers whereas our decomposition approach needs only 256 180-bit blocks. Besides the half-size multipliers, an adder with a long carry chain is also required to merge the outputs from two smaller multipliers. For instance, the last stage of the 46080×180-bit multiplication needs a 23040-bit + 180-bit carry-propagation adder. The carry propagation chain can be pipelined by dividing the large merging adder into multiple stages. The pipeline dataflow is designed such that idle small adders can be reused to minimize the number of arithmetic units required. Depending on the resource available in each configurable logic block (CLB), different levels of time multiplexing can be used to adapt to the heterogeneity of FPGA devices. For the Ultrascale+ architecture, we exploit 4:1 multiplexing to reuse each small adders four times, as the MUX logic can be optimally fitted into a single CLB associated with the CARRY8 arithmetic component without extra resource occupation. The schematic of these time-multiplexing chain adders is shown in Fig. 2(a). Other than the arithmetic processing, logical operations on large integers are also resource hungry. While multiple-input LUTs in the CLB have often been used for the implementation of complicated logic functions like wide-input multiplexer on FPGA, they are inefficient when the bus width is high. To allow logical operations to be processed efficiently at different levels of decomposition, our proposed design utilizes the hard-wired DSP blocks available on the FPGA device to realize large logical operators (e.g., wide-bus multiplexing) on large integers. Fig. 2(b) shows the configuration of the DSP for high-width operands. This has eliminated the large consumption and inefficient use of CLBs for wide-bus operations.

Fig. 3 illustrates the complete architecture of the FAHE encryption processor. It consists of three main units, a large multiplier, a control unit (FSM) and a random number generator (RNG). Upon asymmetric decomposition, the symmetric 180-bit multiplier is a fundamental reusable arithmetic unit.

Algorithm 1: Table-lookup-based modular reduction

1 **Input**: Modulus n, Large integer z, k=BitLength(n), m=BitLength(z), Precomputed lookup table T
2 **Output**: z mod n.
3 Initialize: $s \leftarrow Binary(z)$, $r \leftarrow 0$;
4 **for** $i \leftarrow m-1$ *to* k **do**
5 | **if** $s[i] = 1$ **then**
6 | | $r \leftarrow r + T[i]$;
7 | **end**
8 **end**
9 $r \leftarrow r + \Sigma_{j=0}^{k-1} s[j]2^j$
10 **while** $r \geq n$ **do**
11 | $r \leftarrow r - n$
12 **end**
13 **if** $r < 0$ **then**
14 | $r \leftarrow r + n$
15 **end**
16 return r

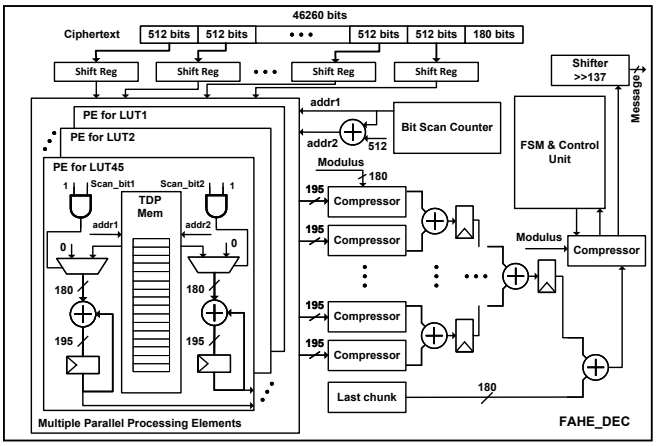

Fig. 4. Architecture of the FAHE decryption processor.

Our strategy is to design a compact 180-bit multiplier that can also allow multiple such basic computing blocks to be instantiated for a larger bit-width multiplication in case the large multiplication cannot be fully decomposed into 180-bit multiplications on certain FPGA devices. We choose a 180-bit sequential Booth encoded multiplier for an area-efficient FPGA implementation. It can realize a digit-serial multiplication with a smaller number of partial products to be accumulated using only a single carry-propagation adder. Different encryption key values can be flexibly processed by reloading the encoded parameters. To minimize side-channel leakage, a constant time implementation of the Booth multiplier is adopted in which the number of partial products and the partial product accumulation time are fixed for any inputs. With the time multiplexing of long pipeline adders and the large bitwidth (23040-bit and 11520-bit) multiplexers implemented using DSP blocks, we can fully decompose a 11520-bit multiplier down to 64 180-bit symmetric multipliers and implement these 180-bit multipliers using the constant-time Booth multiplier on the ZCU102 FPGA board. This 11520-bit block is then reused to construct the 46080-bit asymmetric multiplier. The FSMs shown in Fig. 3 are introduced to schedule the data flow for reusing the 11520-bit multiplier for larger operands. According to (3), to complete the encryption, an extra pipelined adder is needed to add the message m with noise to the product $p \cdot q$. Taking into consideration the resources and maximum frequency of the FPGA, the random number is generated using a lightweight cryptographically strong pseudo-random number generator (CPRNG) with maximum length sequence instead of a true random number generator. The seed value is kept secret and changed regularly or obtained from the random metastability outputs (e.g., by overclocking). Each CPRNG is implemented using a Linear Feedback Shift Register (LFSR).

B. FAHE Decryption

The main computational bottleneck of the FAHE decryption is the modular reduction for ciphertext modulo p, where the ciphertext is >45714 bits after aggregation and p is 178 bits (same as the encryption). A larger design is made to support ciphertexts of up to 46260 bits and 180-bit p. In Paillier accelerators, modular exponentiation is accomplished by iterative modular multiplications. A Montgomery modular multiplication [11] transforms the expensive arbitrary integer modulo reduction to a faster modulo power-of-two integer reduction to enable successive modular multiplications to be performed efficiently within the Montgomery domain. Another popular modular reduction approach is Barrett algorithm [26], which converts the division into a multiplication with precomputed coefficient. When the bit length of the input is comparable to the bit length of the modulus, Barrett reduction is more efficient. Since the ciphertext is 46260 bits and p is 180 bits, and FAHE does not require modular exponentiation, both approaches have their limitations.

Instead of Montgomery and Barrett methods, Algorithm 1 illustrates a fast modular reduction method [18] based on a table-look-up approach. It converts the division into a series of additions. Although both Paillier and FAHE decryptions require modular reduction, the table-look-up method has some constraints for Paillier. The bitwidth of the accumulator in line 6 depends on the size of the modulus, but Paillier requires 6144-bit modulus for the same level of security as RSA-3072. The data dependency of the for loop (lines 4-8) shaded in pink color and the while loop (lines 10-12) shaded in cyan color severely constrains the pipelinability the 6144-bit adder and limits the maximum operating frequency of the entire decryption process. By contrast, the operand size of the accumulation in FAHE decryption is only 180 bits. In addition, the bit scanning (line 5) and table lookup operation have no data dependency. Hence, the for loop can be unrolled for parallel computation. In our proposed design, we tailor the degree of parallelism to the available options of on-chip memory. For the Xilinx Ultrascale+ family, each block RAM (BRAM) instance can store up to 36 kbits of data. It supports the configuration of 1k (depth)$\times$36 bits or 512×72 bits. Using the 72-bit option will lead to a waste of the full BRAM capacity. This is because each entry of our lookup table is a 180-bit integer, which needs 2.5 (i.e, 3) BRAM instances. Half of the space of the last

TABLE I
IMPLEMENTATION RESULTS OF THE PROPOSED FAHE PROCESSORS ON ZCU102

Design	CLB	LUT	Reg	Carry8	DSP	BRAM
FAHE Encryption Processor						
DP	33270	137029	187878	16317	0	0
DP+TM	27957	138388	151308	7301	0	0
DP+TM+WB	26834	139305	151540	7301	840	0
64FP	**33487**	**226025**	**191740**	**10337**	**720**	**0**
Usage %	**97.74%**	**82.47%**	**34.98%**	**30.17%**	**28.57%**	**0%**
FAHE Decryption Processor						
45PE	**10846**	**73722**	**41611**	**8250**	**0**	**225**
Usage %	**31.66%**	**26.9%**	**7.59%**	**24.08%**	**0%**	**24.67%**

TABLE II
PERFORMANCE COMPARISON ON HARDWARE AND SOFTWARE IMPLEMENTATIONS OF ADDITIVE HES

Design	Yang et al [12]	Bahadori et al [13]	San et al [14]	PipeFL [15]	FAHE SW [17]	**This Work**
Platform	AWS F1	zu9eg	7vx330t	GX2800	i7-7700K	**zu9eg**
Algorithm	Paillier	Paillier	Paillier	Paillier	FAHE	**FAHE**
Security	128 (3072)	112 (2048)	112 (2048)	≤ 80 (1024)	128 (3072)	**128**
Encryption Process						
Fmax (MHz)	500	193	325	187	4200	**150**
# cycles	n/a	$\sim$2.67M	$\sim$34M	125,663	330,027	**739**
OP/s	$<$500	72.3	9.5	1488.1	12,726	**202,977**
Decryption Process						
Fmax (MHz)	500	193	312	187	4200	**200**
# cycles	n/a	$\sim$2.67M	$\sim$34M	125,663	293,469	**1125**
OP/s	$<$500	72.3	9.1	1488.1	14,311	**177,777**

BRAM instance is empty. By using the 36-bit configuration, the 180-bit data can be mapped to exactly 5 BRAM instances with no wastage. The complete lookup table is partitioned into 45 subtables, each stores 180 (36×5) kbits. This allows 45 accumulators to concurrently accumulate the values fetched from these subtables independently. To further reduce the latency, we enable the true dual port (TDP) interface of the BRAM to allow two read access ports to fetch two values from the memory in the same clock cycle. Two accumulators are used in one PE associated with one subtable. Thus, each PE can process two chunks (each chunk with 512 bits) of ciphertext simultaneously, with the address to the subtable for the second chunk of each PE offset by 512. After bit scanning, the 90 accumulated intermediate results are sent to 90 compressors, each contains one comparator and one subtractor. For constant time implementation, a maximum of 511 cycles is required for each individual compressor to subtract p from the intermediate result if it is larger than p. A 7-stage adder tree is utilized to sum all the 90 intermediate results and the last chunk of ciphertext together. Another compressor with 90-cycle latency is introduced to subtract p from the result of the adder tree in each cycle if it is larger than p. The decrypted message is obtained by right shifting the reduced result by 137 ($\rho+\alpha$) bits. The overall architecture of the decryption processor is shown in Fig. 4.

IV. EXPERIMENTAL RESULTS

The proposed FAHE encryption and decryption processors are implemented and evaluated on the Xilinx ZCU102 development board with XCZU9EG-2FFVB1156 FPGA device. Our designs are described in Verilog in the Vivado 2022.1 environment. Test circuit is included with the design to physically verify the functionality of our designs on the real silicon. The golden test vectors are generated in Python.

A. Hardware Implementation Results

Table I summarizes the hardware implementation results of FAHE encryption processor and decryption processor. The encryption processor is designed for 150MHz clock frequency. The asymmetric decomposition (DP) is the main contributor that makes it feasible to map the large multiplier of the FAHE encryption onto this device. Without additional resource optimization effort, we can decompose a 5760-bit multiplier down to 32 180-bit symmetric multipliers. The 180-bit multiplier is the fundamental processing (FP) block in our design. The fully decomposed 5760-bit multiplier is reused to construct the 46080-bit asymmetric multiplier. The DP design consumes 32.5k CLBs and 16k Carry8 units. Time multiplexing (TM) allows reuse of adders as shown in Fig. 2(a). The Carry8 usage of DP+TM design is reduced by more than 50% compared to DP, which amounts to a reduction of 5k CLBs. It is worth noting that the 4:1 time multiplexing can reuse the small adders four times, but those Carry8 units used in the FP blocks are not reusable with time multiplexing. By using the DSPs as wide-bus multiplexers, DP+TM+WB design further increases the area efficiency. Although the CLB usage is similar to DP+TM, the congestion level of the entire FAHE encryption processor has reduced significantly during placement and routing. This makes it possible to decompose a 11520-bit multiplier instead of only a 5760-bit multiplier. Thus, we have implemented the 64FP DP+TM+WB version. Its results are printed in boldface in the last two rows of the FAHE encryption processor resource utilization of Table I. The 64FP design incorporates 64 180-bit symmetric Booth encoded sequential multipliers, which improves the throughput by 41% compared to the 32FP version. For the FAHE decryption processor, we implement the 45PE design at 200MHz. The ciphertext will be split into 90 chunks for parallel computation as mentioned in Sec. III.B. It consumes 26.9%, 7.59%, 24.08% and 24.67% of LUT, Reg, Carry8 and BRAM resources, respectively. The BRAMs are consumed by the 45 ($1k \times 180$)-bit subtables. The configured BRAM space is fully utilized with 100% utilization rate.

B. Comparison Results on Performance

We compare the performances of the latest software (CPU) implementation of FAHE, and the state-of-the-art hardware implementations of Paillier against our FAHE encryption processor in Table II. As reported in [17], FAHE has $121\times$ speedup compared to Paillier in software implementation with the same security level. The latency and throughput reported for our 64FP design, software implementations and [12] are based on 128-bit security strength. Other reported results are based on lower security strength. Among all the existing Paillier hardware accelerators, the design of [12] has the highest throughput of up to 500 encryptions per second (OP/s). We would like to highlight that the 1,488.1 OP/s reported in PipeFL [15] and

5,200 OP/s reported in [12] for Paillier encryption were both based on the key size for 80-bit security level only. Compared to [12], our FAHE processor can make 202,977 encryptions per second, which gives a speedup of no less than 404×. Compared to the best FAHE software implementation on the 4.2GHz CPU, our design has increased the throughput by 14.95×.

The decryption throughput of existing software and hardware implementations of Paillier are similar to their encryption as shown in Table II. Our FAHE decryption processor outperforms the highest throughput achievable by existing Paillier hardware implementation [12] by at least 354×, yet our design requires less than 32% of hardware resources on the ZU9EG FPGA device while [12] has almost exhausted all the resources on the VU9P (which has ∼3× more LUTs than ZU9EG) FPGA chip. Our decryption processor outruns the FAHE software decryption with 11.49× speedup in throughput rate. Our decryption processor can run at 200MHz with the current parallel PE configuration, and each decryption can be completed in 1125 clock cycles. The computing parallelism can be further increased by splitting the large lookup table to more subtables at the expense of reducing the utilization rate of each BRAM instance and requiring more compressors.

V. Conclusion

We have presented the first-ever attempt to design a resource-efficient and high-throughput FPGA-based cryptoprocessor to accelerate FAHE algorithm for privacy-preserving secure aggregation in cross-silo FL. A small symmetric area- and latency-efficient Booth-encoded digit-serial multiplier has been utilized as a basic computing unit to construct a much larger integer multiplier of >40kbits. To avoid exploding the FPGA resources, our proposed encryption processor architecture features asymmetric multiplier decomposition, reuse of small chain adders by time multiplexing and DSP-based wide-bus multiplexers. The decryption hardware is designed by dividing the operands of the modulo operation into multiple chunks and partition a large pre-computed lookup table into a number of smaller subtables commensurate with the number of PEs for concurrent computation. Our proposed design can support 512 homomorphic additions with 128-bit security strength. It increases the throughput of state-of-the-art Paillier accelerators for the same application by more than 404× for encryption and more than 354× for decryption on the ZCU102 platform. Compared to FAHE software-based implementation, our design achieves 14.95× and 11.42× higher throughput for encryption and decryption, respectively.

FAHE1 algorithm [17] expands the ciphertext by 6.5× of Paillier. Although communication bandwidth is not a major limitation in the cross-silo FL compared with the cross-device FL, it would be good to reduce the ciphertext size. One possible future work is to consider efficient FPGA implementation of FAHE2 algorithm, which reduces the ciphertext size to half of FAHE1 but has a stronger security assumption. Batching multiple messages [10] is another potential approach to reduce the amount of data transmission in FL.

References

[1] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. a. Ranzato, A. Senior, P. Tucker, K. Yang, Q. Le, and A. Ng, "Large scale distributed deep networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 25, Lake Tahoe, Nevada, USA, Dec. 2012.

[2] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Security*, Denver, Colorado, USA, 2015, p. 1310–1321.

[3] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proc. 20th Int. Conf. Artificial Intell. Statistics*, vol. 54, Florida, USA, 20–22 Apr 2017, pp. 1273–1282.

[4] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *Proc. IEEE Symp. Secur. Priva. (SP)*, San Francisco, CA, USA, Sep. 2019, pp. 691–706.

[5] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, "Learning differentially private recurrent language models," presented at Int. Conf. Learning Representations (ICLR), 2018.

[6] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 5, pp. 1333–1345, May 2018.

[7] C. Huang, J. Huang, and X. Liu, "Cross-silo federated learning: Challenges and opportunities," *arXiv preprint arXiv:2206.12949*, 2022.

[8] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st Ann. ACM Symp. Theory Computing*, ser. STOC '09, Bethesda, MD, USA, 2009, p. 169–178.

[9] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. Advances Cryptology — EUROCRYPT '99*, Berlin, Heidelberg, 1999, pp. 223–238.

[10] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, "BatchCrypt: Efficient homomorphic encryption for Cross-Silo federated learning," in *Proc. 2020 USENIX Conf. Usenix Ann. Tech. Conf.*, ser. USENIX ATC'20, Jul. 2020, pp. 493–506.

[11] P. L. Montgomery, "Modular multiplication without trial division," *Math. Computation*, vol. 44, no. 170, pp. 519–521, 1985.

[12] Z. Yang, S. Hu, and K. Chen, "Fpga-based hardware accelerator of homomorphic encryption for efficient federated learning," *arXiv preprint arXiv:2007.10560*, 2020.

[13] M. Bahadori and K. Järvinen, "A programmable soc-based accelerator for privacy-enhancing technologies and functional encryption," *IEEE Trans. VLSI Syst.*, vol. 28, no. 10, pp. 2182–2195, Jul. 2020.

[14] I. San, N. At, I. Yakut, and H. Polat, "Efficient paillier cryptoprocessor for privacy-preserving data mining," *Sec. and Commun. Netw.*, vol. 9, no. 11, pp. 1535–1546, Jul. 2016.

[15] Z. Wang, B. Che, L. Guo, Y. Du, Y. Chen, J. Zhao, and W. He, "Pipefl: Hardware/software co-design of an fpga accelerator for federated learning," *IEEE Access*, vol. 10, pp. 98 649–98 661, Sep. 2022.

[16] E. Barker, "Recommendation for key management: Part 1—general (nist special publication 800-57 part 1 revision 5)," *National Institute of Standards and Technology (NIST): Gaithersburg, MD, USA*, 2020.

[17] E. L. Cominetti and M. A. Simplicio, "Fast additive partially homomorphic encryption from the approximate common divisor problem," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 2988–2998, Apr. 2020.

[18] Z. Cao, R. Wei, and X. Lin, "A fast modular reduction method," *Cryptology ePrint Archive*, 2014.

[19] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Process. Mag.*, vol. 37, no. 3, pp. 50–60, 2020.

[20] L. Jiang, Q. Lou, and N. Joshi, "Matcha: A fast and energy-efficient accelerator for fully homomorphic encryption over the torus," in *Proc. 59th ACM/IEEE Des. Auto. Conf. (DAC)*, San Francisco, California, 2022, p. 235–240.

[21] "Fate (federated ai technology enabler)," https://github.com/FederatedAI/FATE.

[22] C. Data61, "Python paillier library," https://github.com/data61/python-paillier, 2013.

[23] S. Kim and J. Jin, "Intel paillier cryptosystem library," https://github.com/intel/pailliercryptolib.

[24] C. Rafferty, M. O'Neill, and N. Hanley, "Evaluation of large integer multiplication methods on hardware," *IEEE Trans. Comput.*, vol. 66, no. 8, pp. 1369–1382, Aug. 2017.

[25] E. Ustun, I. San, J. Yin, C. Yu, and Z. Zhang, "Impress: Large integer multiplication expression rewriting for fpga hls," in *Proc. IEEE 30th Ann. Int. Symp. Field-Programmable Custom Computing Machines (FCCM)*, New York City, NY, USA, Jun. 2022, pp. 1–10.

[26] P. Barrett, "Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor," in *Advances in Cryptology — CRYPTO' 86*, Berlin, Heidelberg, 1987, pp. 311–323.