RACE: RISC-V SoC for En/decryption Acceleration on the Edge for Homomorphic Computation

Zahra Azad, Guowei Yang, Rashmi Agrawal
Boston University, USA
{zazad,guowei,rashmi23}@bu.edu

Daniel Petrisko, Michael Taylor
University of Washington, USA
{petrisko,profmbt}@cs.washington.edu

Ajay Joshi
Boston University, USA
joshi@bu.edu

ABSTRACT
As more and more edge devices connect to the cloud to use its storage and compute capabilities, they bring in security and data privacy concerns. Homomorphic Encryption (HE) is a promising solution to maintain data privacy by enabling computations on the encrypted user data in the cloud. While there has been a lot of work on accelerating HE computation in the cloud, little attention has been paid to optimize the en/decryption on the edge. Therefore, in this paper, we present RACE, a custom-designed area- and energy-efficient SoC for en/decryption of data for HE. Owing to similar operations in en/decryption, RACE unifies the en/decryption datapath to save area. RACE efficiently exploits techniques like memory reuse and data reordering to utilize minimal amount of on-chip memory. We evaluate RACE using a complete RTL design containing a RISC-V processor and our unified accelerator. Our analysis shows that, for the end-to-end en/decryption, using RACE leads to, on average, 48x to 39729x (for a wide range of security parameters) more energy-efficient solution than purely using a processor.

ACM Reference Format:

1 INTRODUCTION
Over the last decade, Homomorphic Encryption (HE) has emerged as one of the key techniques to perform privacy-preserving computations. Edge devices (having energy and area constraints) can therefore leverage cloud services to compute on private user data using HE. Figure 1 shows an example of HE-based computing where a user captures a picture/video using an edge device, pre-processes it, encrypts it, and then sends it to the cloud for further processing. The cloud operates on the encrypted data and sends the encrypted result back to the user, and only the user can decrypt the result.

Several recent works have focused on accelerating the HE operations on the cloud through algorithmic optimizations for CPU [4, 10]. GPU [1, 9], and custom hardware accelerators [7, 16, 17, 21]. All these works make an implicit assumption that the edge-side operations, including encryption and decryption (en/decryption), encoding and decoding (en/decoding), and error sampling are trivial and do not need to be accelerated. However, these edge-side operations are not trivial, and have high compute and memory requirements. SEAL-Embedded is the first HE library targeted for embedded devices, which employs several computational and algorithmic optimizations to achieve memory efficient en/decoding and en/decryption on edge devices [14]. They target Cheon, Kim and Song (CKKS) [6] HE scheme as it operates on floating-point data. However, the memory efficient implementation of the library has performance bottlenecks (e.g., inefficient modular arithmetic implementation) and is still not practical. For example, if we were to encrypt video captured by a QQVGA camera operating at a low resolution of 120 × 160 pixels using memory-efficient SEAL-Embedded library (running at 1 GHz on a RISC-V core like BlackParrot (BP) [15] for polynomial degree of $N = 4096$ and three 30-bit primes), we cannot encrypt even one frame per second (more details in Section 5). Typically surveillance cameras and mobile platforms (forming the ‘Internet of Video Things’) have an average frame rate of 15 and 30 frames per second [22]. One could use a more powerful processor, but then the power consumption would be higher which would not be sustainable in a typical edge device.

The key bottleneck of the edge-side operations is the en/decryption operation, where in the main bottleneck is the Number Theoretic Transform (NTT) operation. Several prior works have accelerated the NTT operation in the context of Post Quantum Cryptography (PQC) [2, 5, 11, 13, 23]. However, the parameters used by these works are much smaller (polynomial degree $N < 2^{10}$ and coefficient bit width $\log Q < 24$) than the required parameters for a practical HE application ($N > 2^{12}$ and $\log Q > 109$). None of these NTT accelerators focus on designing an area and energy-efficient solution for en/decryption to support HE-based computing. Su et al., proposed an FPGA [19] accelerator for en/decryption targeting Brakerski-Gentry-Vaikuntanathan (BGV) HE scheme [8], but it supports small security parameters ($N = 2^7$). Similarly, Yoon et al., also proposed an ASIC based en/decryption accelerator [24], but it is also evaluated only for small security parameters ($N = 2^5$).

In this work, we present RACE: a custom-designed area- and energy-efficient RISC-V System-on-Chip (SoC) for en/decryption of the data on the edge. Encryption and decryption perform similar operations (polynomial addition and multiplication),
and so we propose a unified accelerator, where the encryption and decryption operations share the datapath. To reduce SRAM area in RACE we architect it such that it requires memory that is large enough to only store two polynomials. This memory is reused over time to store inputs, outputs, and intermediate values. We propose a novel data reordering scheme for NTT so that RACE where each element is chosen from a vector and returns an integer polynomial by computing:

\[ c = m \cdot \text{rk}_{2} \]

Considering our example Video Frame Encryption Example: Using Quarter Quarter VGA (QQVGA) native plaintext data-type in CKKS scheme is a vector of length \( 2 \) and so we use Equation (1) and (2) for encryption. Both the equations compute the ciphertext: While polynomial addition is straightforward, NTT is commonly used to speedup polynomial multiplication. We split the operations on \( (p_{k0}, \mu, m + e_{0}) \) to calculate \( c_{0} \) and operations on \( (p_{k1}, \mu, e_{1}) \) to calculate \( c_{1} \) into two ‘half-encryption’ operations. Thus, every encryption operation calls the accelerator twice, once for \( c_{0} \) half-encryption and once for \( c_{1} \) half-encryption.

2 PRELIMINARIES

CKKS Scheme - En/Decryption Operations: The CKKS HE scheme can perform computations on encrypted real numbers. The native plaintext data-type in CKKS scheme is a vector of length \( N/2 \) where each element is chosen from \( \mathbb{C} \), the field of complex numbers. The encoding operation takes as input this \( N/2 \)-dimensional vector and returns an integer polynomial \( m(x) \). Encryption of the polynomial \( m(x) \) under the public key \( pk \) generates a ciphertext \( ct \) by computing:

\[ c_{0} = \mu \cdot p_{k0} + m + e_{0}, \]
\[ c_{1} = \mu \cdot p_{k1} + e_{1} \]

Here, \( \mu \) is a uniformly sampled polynomial, and \( e_{0} \) and \( e_{1} \) are two polynomials sampled from a discrete Gaussian noise sampler. The coefficients in both the polynomials \( (c_{0}, c_{1}) \) are elements of \( \mathbb{Z}_{Q} \), where \( Q \) is typically on the order of thousands of bits to account for the noise growth. Therefore, to compute on such large operands efficiently, the CKKS scheme supports the use of Residue Number System (RNS) (also known as the Chinese Remainder Theorem (CRT) representation). Using this approach, each number \( x \) is represented modulo \( Q = \prod_{i=1}^{\ell} q_{i} \), where each \( q_{i} \) is a prime number. We can represent \( x \in \mathbb{Z}_{Q} \) as a length-\( \ell \) vector of scalars \( [x]_{B} = (x_{1}, x_{2}, \ldots, x_{\ell}) \), where \( x_{i} \equiv x \mod q_{i} \). We refer to each \( x_{i} \) as a limb of \( x \). The ciphertext is decrypted to obtain the original message back using:

\[ m = c_{0} + c_{1} \cdot s \mod q_{\ell} \]

Here \( s \) is the secret key. Using RNS, both en/decryption can be performed w.r.t. a smaller modulus \( q_{i} \) instead of a large modulus \( Q \).

Video Frame Encryption Example: Considering our example of video frame encryption, using Quarter Quarter VGA (QQVGA) frame resolution, the frame size is \( 120 \times 160 \) pixels. If this frame is in grey scale, the frame size will be \( 120 \times 160 \times 8 = 153,600 \) bits = 19.2 KB. With \( N = 4096 \) and \( \log q = 30 \) bits, we can encode \( N/2 \cdot \log q = 2048 \times 30 = 61,440 \) bits in a single ciphertext, which implies that a single frame will be encoded and encrypted within 3 ciphertexts and will have a total size of 327 KB.

BlackParrot RISC-V Multicore: BP is an agile open-source RISC-V multi-core processor for accelerator SoCs [15]. It implements the RISC-V RV64G architecture and is designed as a scalable, configurable heterogeneously tiled micro-architecture. BP provides a robust and scalable end-to-end framework for accelerator integration, which simplifies interfacing both coherent and streaming accelerators, and the offloading of parts of the user application from the processor to the accelerator. This framework provides hardware implementation of accelerator tiles in SystemVerilog and helps accelerator designers and system architects to evaluate their accelerator ideas and evaluate the end-to-end application time.

3 RELATED WORK

One of the key works in the area of accelerating edge-side operations for HE is the Microsoft SEAL-embedded library [14], which focuses on reducing the memory requirement for en/decoding and en/decryption operations. It uses RNS partitioning, data type compression, memory pooling and reuse to reduce the memory consumption. However, this software-based implementation of HE encryption is still slow and not efficient for real-time applications. As mentioned earlier, for a video application with QQVGA resolution, SEAL-embedded fails to encrypt even one frame per second.

There are few works focusing on accelerating en/decryption for HE [19, 24]. Su et al. [19] present an FPGA-based accelerator, but it is for the BGV HE scheme as against the CKKS scheme that we support. Although their accelerator can be extended to larger polynomial degrees to support higher security levels, in its current form it only supports small parameters \( (N = 128, \log Q = 27) \), which are impractical for HE computation. The authors have left the support for larger parameters as part of the future work. Moreover, the accelerator is mainly optimized for higher performance and throughput, and not for area/energy efficiency. Yoon et al. [24] present an ASIC-based en/decryption accelerator for HE operations. The accelerator is again evaluated for small \( N = 16 \) only. It needs large buffers to store the in/outputs and the pre-computed twiddle factors, increasing the memory area.

In our work, we can perform en/decryption for any practical security parameters. We share the datapath, adopt memory reuse and data reordering strategies, and compute all the twiddle factors on-the-fly to enable efficient en/decryption operations.

4 RACE: DESIGN AND FUNCTIONALITY

4.1 Dataflow for Encryption and Decryption

We use Equation (1) and (2) for encryption. Both the equations perform polynomial addition and multiplication operations to compute the ciphertext. While polynomial addition is straightforward, NTT is commonly used to speedup polynomial multiplication. We split the operations on \( (p_{k0}, \mu, m + e_{0}) \) to calculate \( c_{0} \) and operations on \( (p_{k1}, \mu, e_{1}) \) to calculate \( c_{1} \) into two ‘half-encryption’ operations. Thus, every encryption operation calls the accelerator twice, once for \( c_{0} \) half-encryption and once for \( c_{1} \) half-encryption.
The decryption operation follows Equation (3) and accepts inputs in NTT format. At a high-level, the half-encryption and decryption operations perform the same underlying operations, just in a different order. Hence, we share the datapath and control logic between encryption and decryption to lower the accelerator area (see Figure 3).

4.2 RACE System View

RACE SoC consists of a single-core configuration of BP and an en/decryption accelerator. We interface the en/decryption accelerator with BP as a streaming accelerator because it needs a large amount of input and output data transfers (tens of KBs). We set up a hardware DMA logic to transfer all the input data from the main memory to the accelerator tile. The required input data includes \(\text{pk}_0, \text{pk}_1, e_0, e_1\) and \(\mu\). Once the accelerator completes en/decryption operation, it interrupts the BP core to signal a operation completion. Then, the DMA logic transfers the accelerator output data to the main memory that can be read by BP for further processing.

4.3 Accelerator Microarchitecture

Figure 2 shows the detailed microarchitecture of our pipelined accelerator. The Butterfly Unit (BFU) is pipelined and is designed to perform NTT, INTT, polynomial addition and multiplication operations that are required by both encryption and decryption operations (see Figure 3). The permutation unit (PU) reorderers (more details given later in this section) the output generated by the BFU and writes it back into the SRAM banks. The control unit (CU) generates activation signals for different datapaths corresponding to the different operations.

### 4.3.1 BFU

A Butterfly operation (BF) is the building block of NTT/INTT. An NTT/INTT operation consists of \(\log_2 N\) stages \((N = 1\) is the polynomial degree), and each stage contains \(N/2\) BFUs. Given a polynomial \(a\), a BF takes its two coefficients \((a_1, a_2)\) as input and computes \((a_i, a_j) = (a_i + \omega \cdot a_j \pmod{d}, a_i - \omega \cdot a_j \pmod{d})\) (refer Algorithm 1 line 13 and 14). Here, \(\omega\) is the twiddle factor. A degree \(N - 1\) polynomial requires \(N/2\) twiddle factors, where each twiddle factor needs \(\log(d)\) bits. To reduce the memory overhead for storing pre-computed twiddle factors, our accelerator computes them on-the-fly within BFU. BFU is a fully-pipelined module with the throughput of 1 BF per cycle. It contains a modular multiplier where modular reduction operation is performed using a Barrett reduction [3] unit. BFU also has an integer adder and subtractor unit that performs modular reduction using conditional operator. The latency of the pipelined modular multiplier can be tuned through the number of pipeline stages. The multiplier lies on the critical path in the accelerator, and we pipeline the multiplier to improve the frequency of the accelerator. As power and area are the primary design goals for embedded devices, all the above computations are performed by sequentially leveraging the pipelined BFU.

### 4.3.2 SRAM Arrays

We use the SRAM arrays to store the input/output and intermediate polynomials. We propose two ideas: memory reuse and reordering to minimize the SRAM size.

**Memory reuse:** For efficient en/decryption computation, all the required polynomials \((m, e_0, e_1, \mu, \text{pk}_0, \text{pk}_1, c_0, c_1)\) should be stored in the on-chip memory of the accelerator. However, a single polynomial is usually large and requires large amount of memory. For \(N = \log_2\) 4, \(\log(d) = 3\) we need 480 KB to store all the in/out polynomials. In our memory reuse approach, we manage the en/decryption operations such that at any point of time we need to store at most two polynomials, which require 122 KB space. We divide the on-chip SRAM memory into multiple banks. Each polynomial is stored across multiple banks and those banks together form a group. We have two bank groups i.e., BG0 and BG1 for the two polynomials. These bank groups are used for storing the input/output and intermediate polynomials during en/decryption operation. Figure 4 (a) and (b) show how the two bank groups are shared among the various polynomials during encryption and decryption operation, respectively. For example, we perform an in-place NTT/INTT operation that reads the data for polynomial \(\mu\) from BG0, operates on it, and writes the results back BG0. While computing NTT on the polynomial \(\mu\), we load the next input polynomial \(\text{pk}_1\) into BG1 in parallel. We perform memory re-use during the modular addition and multiplication operations as well. Both of these operations read inputs from bank groups BG0 and BG1 and write the results back to BG1 only. So we can reuse BG0 for the next operation once the modular addition or multiplication operations are finished. Thus, through memory reuse approach, we can perform en/decryption efficiently using a small memory that stores only two polynomials.

**Memory reordering:** A naive implementation of the NTT algorithm requires 2 read and 2 write port (2R2W) memory bank of size \(N\) to
Algorithm 1: NTT_swap

Input: Polynomial $a(x) \in \mathbb{Z}_q[x]$ in bit-reversed order
Output: $\text{NTT}(a(x))$ in normal order

1: $m = 2$
2: for $(\text{stage} = 0; \text{stage} < (\log N - 1); \text{stage}++)$ do
3:   $\omega = 1; \omega_m = \omega_2^{\log N-1-\text{stage}}; \text{upd_cnt} = 1$
4:   for $(j = 0; j < m * 2; j++)$ do
5:     for $(k = 0; k < N; k++ = m * 4)$ do
6:       $i0 = [\lfloor i \rfloor];$
7:       for $(l = 0; l < 4; l++)$ do
8:         switch $l$ do
9:         case 0 do $id = j + k$
10:            case 1 do $id = j + k + 2$
11:            case 2 do $id = j + k + m$
12:            case 3 do $id = j + k + m * 2$
13:       $a[id] = a[id] + a[id + 1] * \omega (\text{mod } q)$
14:       if $\text{upd_cnt} = 2^\text{stage} + 1$ then
15:         $\omega = \omega \times \omega_m (\text{mod } q); \text{upd_cnt} = 1$
16:       else $\text{upd_cnt} = \text{upd_cnt} + 1$
17:   $m = (m == N / 4) ? 2 : (m * 2)$
20: end for
21: end for
22: /* Bit manipulation */
23: $\text{phy_addr} = \{i[\log N - 3 : 2], i[\log N - 1 : \log N - 2], i[1 : 0]\}$
24: $a_{out}[i] = a_{\text{phy_addr}}$
25: return $a_{out}$

Figure 4: Memory reuse during (a) encryption and (b) decryption operations. "Read/Operate/Write" means the bank group is being accessed during the operations. "Occupied" means the bank group stores intermediate results.

Figure 5: NTT_swap with $N = 32$. The red colored numbers denote the order of BF operations. The consecutive four BFs (2 rows) being reordered are denoted with the same color.
works for both NTT and INTT operations. Based on the mode signal, the PU will be active only during the NTT/INTT computations.

4.3.4 Control Unit. The CU consists of two components – the computation controller and the I/O controller. The computation controller is an FSM that determines the BFU and PU mode signals depending upon the current operation (NTT/INTT, modular addition and multiplications). It also generates the read/write addresses and enable signals for SRAM accesses. During NTT/INTT operation, the computation controller is also responsible for stalling the BFU pipeline and configuring it to compute the twiddle factors on-the-fly. The I/O controller selects the required set of BFU operations depending on the encryption or decryption request received by the accelerator. In addition, it also sets up the DMA unit for the in/output data transfer to/from SRAM arrays.

5 EVALUATION

5.1 Methodology

For our analysis, we run all the edge-side operations from the SEAL-Embedded library on the ‘BP only’ system and RACE in bare-metal mode. In the ‘BP only’ system, we perform all operations on the BP processor. In RACE, we perform the en/decryption operation using the accelerator and the remaining operations on the BP processor. We modified SEAL-Embedded library to execute en/decryption operations on the accelerator in RACE. For both ‘BP only’ and RACE, we use BP SoC with a single core configuration (32 KB each of Icache and Dcache) running at 1 GHz. Both ‘BP only’ and RACE are implemented in SystemVerilog and simulated using VCS. The hardware implementation is cycle-accurate and captures the nuances of data movement between all parts of the systems. For power, performance and area evaluation, we use GlobalFoundries 12 nm technology. We synthesize the logic components in both ‘BP only’ and RACE using Synopsys Design Compiler, and use memory compiler for designing the SRAM arrays.

5.2 Results

Performance: Figure 6 (a) shows the initial setup, encoding, error sampling, DMA, and encryption latency (in clock cycles) for the ‘BP only’ system and RACE for different security parameters (N, log Q). Note that the Y-axis uses a log scale. Similarly, Figure 6 (b) shows the latency breakdown for the initial setup, decode, decrypt and DMA operations. For the ‘BP only’ system, the en/decryption operations take the longest time because they need multiple polynomial multiplications, where the runtime is dominated by NTT/INTT operations. RACE reduces the NTT/INTT execution time by 78.4× for the smallest N (1024) and 121.8× for the largest N (16384). As a result, the encryption time decreases by 62.56-515.45× (80.63-669.56× w/o considering the DMA overhead) and the decryption time decreases by 126.51-160.9× (158.14-201.12× again w/o considering the DMA overhead), which in turn decreases the end-to-end latency by 7.5-312.1× and 9.3-69.5×, respectively. The end-to-end performance improvement is lower than that of en/decryption alone because all the initial setup, encoding/decoding and error sampling operations take non-trivial amount of time and are performed in the software.

In Figure 6 (a) and (b), we observe that we get a higher performance improvement for the larger N values. This is because for larger N values we need to perform more BF operations within an NTT and INTT, and we accelerate these very BF operations using hardware. Moreover, as log Q increases, the number of 30-bit co-primes that we need also increases, which in turn increases the number of times we need to call the encryption and decryption operations (once per co-prime). It is worth noting that for RACE we need to perform DMA operations, but due to the high computational requirements of the en/decryption operations, the DMA overhead is negligible (<20%).

Power/Energy: The total power consumption for an end-to-end en/decryption in the ‘BP only’ system is 27.19 mW, out of which the SRAM power consumption is 41.49% = 11.4 mW and the digital logic consumes the rest of the power. Overall, the power consumption of RACE is about 25-28% (for a range of security parameters) higher than the ‘BP only’ system for both end-to-end encryption and decryption procedures. The increase in the power consumption is due to 41.92-43.55% power increase in the digital logic and 3.36 – 7.81% power increase in the SRAM.

Table 1 shows the energy consumed in the end-to-end encryption and decryption procedures for different (N, log Q) values when using ‘BP only’ and RACE. Overall, RACE consumes 5.07-242.5× lower energy when running an end-to-end encryption procedure and 6.2-54.02× lower energy when running an end-to-end encryption procedure as compared to the ‘BP only’ system. This is because the performance of RACE is up to 312.1× and 69.5× higher for the end-to-end encryption and decryption procedures, but its power overhead is very small. As discussed earlier, RACE speedup is higher for larger security parameters, but the power consumption increases by only 3% for the largest N value compared to the smallest one. Hence, as the security parameters (N, log Q) grow, the end-to-end energy saving per en/decryption increases.

Energy Efficiency: We use EDP metric to compare the energy efficiency of the ‘BP only’ system and RACE (see Table 1). Overall,
RACE has 38.67-75701.92x lower EDP for an end-to-end encryption procedure and 57.93-3756.25x lower EDP for an end-to-end decryption procedure as compared to the 'BP only' system.

**Area:** Overall, RACE area is larger than the 'BP only' system. SRAMs occupy (75%) of the area in the 'BP only' system, and there is a 11%-100% increase in the SRAM area in RACE as compared to the 'BP only' system. However, note that we reduce the SRAM requirement from 480 KB to 120 KB for the largest N value using techniques discussed in section 4.3.2.

**Video Application Evaluation:** For the video application discussed earlier, Figure 7 shows the maximum frames per second (FPS) that both the 'BP only' system and RACE can sustain for different (N, log Q) values when performing an end-to-end encryption. The encrypted frames are shipped to the cloud using a mid-band 5G network, which offers a balance of speed, capacity, and coverage [20]. As shown in Figure 7 (a), in the regions with maximum bandwidth, mid-band 5G network can transfer up to 111 (QQVGA) and 28 (QVGA) frames per second and in the regions with minimum bandwidth, it can only transfer 12 (QQVGA) and 3 (QVGA) frames per second. The 'BP only' system is capable of encrypting up to 3 QVGA FPS for N values smaller than 2048 (refer Figure 7). However, as we increase N to 4096 or larger values, it cannot encrypt even a single frame per second. On the other hand, for QVGA RACE encrypts ~ 20 FPS for small values of N and 10 FPS for the largest N value (16384). For QVGA resolution, the 'BP only' system cannot encrypt even one FPS for the smallest N value (1024). However, RACE can encrypt 6 and 3 FPS for the smallest and largest N values, respectively. While RACE can support higher FPS than the 'BP only', there is still some headroom in both minimum 5G bandwidth and maximum 5G bandwidth cases. Therefore, as part of the future work, we plan to accelerate the en/decoding and the error sampling operations to fully utilize the frame transfer rate that can be sustained by the mid-band 5G network.

6 CONCLUSION

In this work, we present RACE, a RISC-V based SoC for en/decryption acceleration on the edge to support HE operations in the cloud. RACE implements several optimizations that enable high performance, and area- and energy-efficient end-to-end en/decryption operations. Our analyses show that compared to the 'BP only' system, RACE has higher performance and lower energy consumption. As a result, overall RACE is more energy efficient than the 'BP only' system, and has 38.67-75701.92x lower EDP when running an end-to-end encryption procedure and 57.93-3756.25x lower EDP when running an end-to-end decryption procedure.

ACKNOWLEDGMENTS

This material is based on research sponsored by Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) under agreement number FA8650-18-2-7856. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of AFRL and DARPA or the U.S. Government.

REFERENCES


