



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, $48\times$ to $39729\times$ (for a wide range of security parameters) more energy-efficient solution than purely using a processor.

ACM Reference Format:

Zahra Azad, Guowei Yang, Rashmi Agrawal, Daniel Petrisko, Michael Taylor, and Ajay Joshi. 2022. RACE: RISC-V SoC for En/decryption Acceleration on the Edge for Homomorphic Computation. In *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED '22)*, August 1–3, 2022, Boston, MA, USA. ACM, New York, NY, USA, 6 pages. <https://doi.org/10.1145/3531437.3539725>

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,

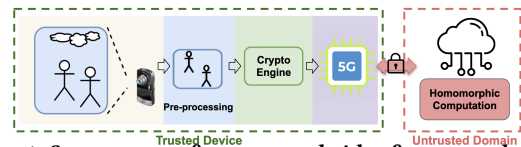


Figure 1: Steps to transfer encrypted video frames to the cloud.

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, 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 $N \leq 2^{10}$ and coefficient bit width $\log Q \leq 24$) 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 ($N = 2^4$).

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),

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ISLPED '22, August 1–3, 2022, Boston, MA, USA

© 2022 Association for Computing Machinery.

ACM ISBN 978-1-4503-9354-6/22/08...\$15.00

<https://doi.org/10.1145/3531437.3539725>

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 only needs single port (1RW) SRAM banks, which further reduces area in contrast with prior works [5, 11–13, 23] that need dual port (1R1W) SRAM banks. We interface our accelerator with BP [15] to design a complete SoC. We provide an end-to-end evaluation of our SoC when performing the en/decryption operation using the accelerator and the remaining operations on the BP processor. We compare our SoC against a ‘BP only’ system where all operations are performed on the BP processor. For both designs we leverage the SEAL-Embedded library. The main contributions of our work are: 1) We profile SEAL-Embedded based edge-side operations for the CKKS scheme on BP, for different N and $\log Q$ pairs to identify the performance bottlenecks; 2) Based on the profiling results, we architect RACE, an area- and energy-efficient SoC to accelerate the en/decryption operations. We use a shared data path for the en/decryption operations, and memory reuse and data reorder techniques; and 3) We interface the accelerator with BP, and evaluate the performance and energy efficiency of RACE when performing end-to-end en/decryption operations. **For the end-to-end encryption and decryption operation, RACE decreases Energy Delay Product (EDP) by 38.67-75701.92 \times and 57.93-3756.25 \times , respectively, compared to the ‘BP only’ system.**

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:

$$c_0 = \mu \cdot pk_0 + m + e_0, \quad (1)$$

$$c_1 = \mu \cdot pk_1 + e_1 \quad (2)$$

Here, μ 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 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- ℓ vector of scalars $[x]_{\mathcal{B}} = (x_1, x_2, \dots, x_{\ell})$, where $x_i \equiv x \pmod{q_i}$. We refer to each x_i as a *limb* of x . The ciphertext is decrypted to obtain the original message back using:

$$m = c_0 + c_1 \cdot s \pmod{q_{\ell}} \quad (3)$$

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 120×160 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 $N/2 \times \log q = 2048 \times 30 = 61,440$ 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 straight-forward, NTT is commonly used to speedup polynomial multiplication. We split the operations on $(pk_0, \mu, m + e_0)$ to calculate c_0 and operations on (pk_1, μ, 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.

Algorithm 1: NTT_swap4

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

```

1  $m = 2;$ 
2 for ( $stage = 0; stage < (\log N - 1); stage + = 1$ ) do
3    $\omega = 1; \omega_m = \omega_n^{2^{\log N - 1 - stage}}; upd\_cnt = 1;$ 
4   for ( $j = 0; j < m * 2; j + = 4$ ) do
5     for ( $k = 0; k < N; k + = m * 4$ ) do
6        $i0 = []; i1 = [];$ 
7       for ( $l = 0; l < 4; l + = 1$ ) do
8         switch  $l$  do
9           case 0 do  $idx = j + k;$ 
10          case 1 do  $idx = j + k + 2;$ 
11          case 2 do  $idx = j + k + m * 2;$ 
12          case 3 do  $idx = j + k + m * 2 + 2;$ 
13           $a[idx] = a[idx] + a[idx + 1] * \omega \pmod q;$ 
14           $a[idx + 1] = a[idx] - a[idx + 1] * \omega \pmod q;$ 
15           $i0.append(idx); i1.append(idx + 1);$ 
16          if  $upd\_cnt == N / (2^{stage+1})$  then
17             $\omega = \omega * \omega_m \pmod q; upd\_cnt = 1;$ 
18          else  $upd\_cnt + = 1;$ 
19           $(a[i0[0]], a[i1[0]], a[i0[1]], a[i1[1]],$ 
20            $a[i0[2]], a[i1[2]], a[i0[3]], a[i1[3]]) =$ 
21            $(a[i0[0]], a[i0[1]], a[i0[2]], a[i0[3]],$ 
22             $a[i1[0]], a[i1[1]], a[i1[2]], a[i1[3]])$ 
23            $m = (m == N/4) ? 2 : (m * 2);$ 
24 for ( $i = 0; i < N; i + = 1$ ) do
25    $/*$  Bit manipulation  $*/$ 
26    $phy\_addr = \{i \lfloor \log N - 3 : 2, i \lfloor \log N - 1 : \log N - 2, i \lfloor 1 : 0\}$ 
27    $;$ 
28    $a\_out[i] = a[phy\_addr];$ 
29 return  $a\_out;$ 

```

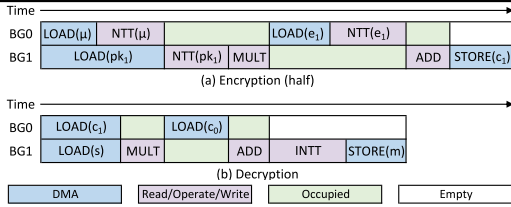


Figure 4: Memory reuse during (a) encryption and (b) decryption operations. “Read/Operate/Write” means the bank group is being accessed during the operations. “Occupied” means the bank group stores intermediate results.

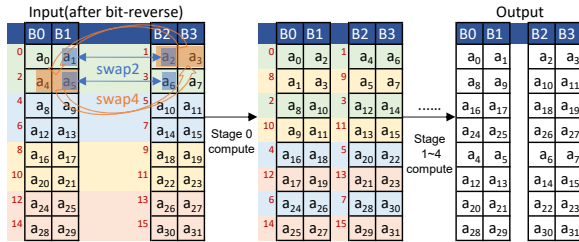


Figure 5: NTT_swap4 with $N = 32$. The red colored numbers denote the order of BF operations. The consecutive four BFs (2 rows) being reordered are denoted with the same color.

achieve two reads and two writes per BF. 2R2W banks are almost $2\times$ larger than 1 read and 1 write port (1R1W) bank. Therefore, just by replacing a single 2R2W bank of size N with two 1R1W banks of size $N/2$ (as long as there are no bank conflicts), we can save half of the memory area. However, the distance between the two inputs of a BF, $(j - i)$, varies across NTT stages. NTT operation iterates through all values from 1 to $N/2$, so there are bank conflicts in certain stages, making this replacement impossible. Existing works reduce the required memory ports from 2R2W to 1R1W by customizing the NTT algorithm. For example, to use 1R1W memory banks for an NTT, Roy et al. [18] proposed a memory-efficient NTT algorithm. We call their approach as NTT_swap2 algorithm. This algorithm avoids bank conflicts (two 1R1W banks) by reordering the output of the two consecutive BF operations. This ensures the input pair needed by BF operation in the next stage resides in different banks.

While the use of 1R1W memory bank saves half the memory area, it is still not efficient. We propose to replace the two 1R1W banks of size $N/2$ with four 1 read/write port (1RW) banks of size $N/4$ to reduce memory area even further. A 1RW bank is $2\times$ smaller than a 1R1W bank. However, this results in newer bank conflicts that cannot be resolved by the existing NTT_swap2 algorithm. If the same bank receives both read and write requests at the same time, we need to have a write buffer that stores the write requests and waits until there are no incoming reads to opportunistically write back the results. The size of the write buffer depends on the number of cycles where the bank is continuously read and written. If there are $N/4$ continuous read and write accesses to the same bank in one particular stage, then the write buffer needs to be the same size as the banks ($N/4$) to store all the write requests that are overlapping with the read requests to the same bank. To avoid the overhead of this large write buffers, we propose a new NTT algorithm called NTT_swap4 (refer Algorithm 1).

On top of NTT_swap2 i.e., reordering the outputs of two BF operations, NTT_swap4 further reorders the output of four consecutive BFs (Figure 5). This is to make sure that not only the two inputs of all BF operations are stored in different banks (NTT_swap2), but also the inputs of consecutive BFs are stored in different banks (NTT_swap4). In this case, the same bank is not continuously accessed and the write buffer can write back the results immediately in the next cycle. Hence, the write buffer can be as small as one element wide ($\log q$), saving further area. Figure 5 shows an example of NTT_swap4 scheme for $N = 32$. The numbers (in red) before each pair of cells denote the order of BF operations. For example, in stage 0, the first four BFs access the following pairs: $(a_0, a_1), (a_2, a_3), (a_4, a_5), (a_6, a_7)$. However, stage 1 expects elements in the order of $(a_0, a_2), (a_4, a_6), (a_1, a_3), (a_5, a_7)$. So we reorder the outputs of stage 0 to the order expected by stage 1 to make sure that the consecutive BFs in stage 1 do not access the same banks for reads and writes (refer Algorithm 1 line 19). We use a PU to perform this reordering.

4.3.3 Permutation Unit. The PU consists of a reordering logic and a small register array to store 8 pairs of BF outputs. Reordering logic starts by writing the two outputs of a BF operation along with their addresses sequentially to the register array in each cycle. After there are eight elements in the register array i.e., four pair of BFU outputs, the reordering logic will first send out the elements stored in even registers and then the elements in odd registers (see memory reorder example in Section 4.3.2). This reordering logic

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 \times for the smallest N (1024) and 121.8 \times for the largest N (16384). As a result, the encryption time decreases by 62.56-515.45 \times (80.63-669.56 \times w/o considering the DMA overhead) and the decryption time decreases by 126.51-160.9 \times (158.14-201.12 \times again w/o considering the DMA overhead), which in turn decreases the end-to-end latency by 7.5-312.1 \times and 9.3-69.5 \times , 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

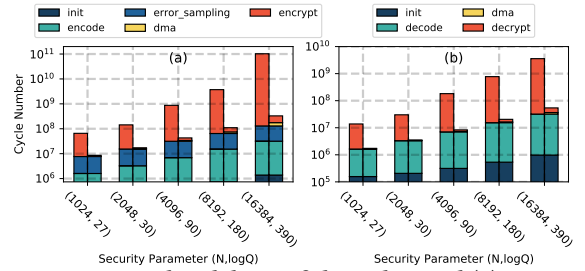


Figure 6: Latency breakdown of the end-to-end (a) encryption, and (b) decryption for ‘BP only’ (left) and RACE (right).

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 \times lower energy when running an end-to-end encryption procedure and 6.2-54.02 \times lower energy when running an end-to-end decryption procedure as compared to the ‘BP only’ system. This is because the performance of RACE is up to 312.1 \times and 69.5 \times 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,

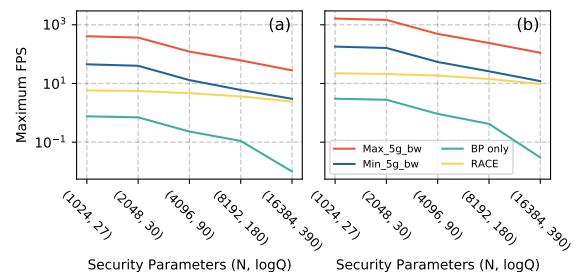


Figure 7: Max. supported (a) QVGA and (b) QQVGA frame rate per second for mid-band 5G, camera, SEAL-Embedded based encryption, and RACE for different N and $\log(Q)$ values.

Table 1: End-to-end performance, power, area and EDP comparison for the ‘BP only’ system and RACE.

$(N, \log Q)$	‘BP only’ (en/decryption)				RACE (en/decryption)			
	Latency (ms)	Energy (mJ)	EDP (mJ.ms)	Area (μ^2)	Latency (ms)	Energy (mJ)	EDP (mJ.ms)	Area (μ^2)
(1024, 27)	65.57/13.8	1.78/0.06	116.91/5.18	135880.17	8.61/1.48	0.35/6.21	3.02/0.09	156901.15
(2048, 30)	143.26/30.09	3.90/0.10	558.10/24.62	135880.17	18.04/3.12	0.58/7.66	10.16/0.33	163856.05
(4096, 90)	871.48/182.16	23.70/0.26	20652.58/902.29	135880.17	42.94/7.6	1.47/19.03	63.17/1.98	175639.99
(8192, 130)	3716.09/774.22	101.05/0.65	375516.07/16299.99	135880.17	110.28/19.08	3.78/32.18	416.93/12.48	199668.39
16384, 390)	102665.07/3576.19	2791.76/1.79	2.86×10^8 /347774.18	135880.17	328.94/51.44	11.51/54.02	3786.11/92.59	250913.93

RACE has 38.67-75701.92 \times lower EDP for an end-to-end encryption procedure and 57.93-3756.25 \times lower EDP for an end-to-end decryption procedure as compared to the ‘BP only’ system.

Area: Overall, RACE area is 15% (smallest N) to 84% (largest N) larger than the ‘BP only’ system area. SRAMs occupy (75%) of the area in the ‘BP only’ system, and there is 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 QQVGA 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 QQVGA 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 38.67-75701.92 \times lower EDP when running an end-to-end encryption procedure and 57.93-3756.25 \times 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

- [1] Ahmad Al Badawi et al. 2021. Multi-GPU Design and Performance Evaluation of Homomorphic Encryption on GPU Clusters. *IEEE TPDS* 32, 2 (2021), 379–391.
- [2] Utsav Banerjee et al. 2019. Sapphire: A Configurable Crypto-Processor for Post-Quantum Lattice-based Protocols. *IACR TCHES* 2019, 4 (2019), 17–61.
- [3] Paul Barrett. 1986. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In *Conference on the Theory and Application of Cryptographic Techniques*. Springer, Springer, Berlin, Heidelberg, 311–323.
- [4] Carl Bootland et al. 2020. Efficiently Processing Complex-Valued Data in Homomorphic Encryption. *Journal of Mathematical Cryptology* 14, 1 (2020), 55–65.
- [5] Xiangren Chen et al. 2022. CFNTT: Scalable Radix-2/4 NTT Multiplication Architecture with an Efficient Conflict-free Memory Mapping Scheme. *IACR CHES* 2022 (2022), 94–126.
- [6] Jung Hee Cheon et al. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *Advances in Cryptology – ASIACRYPT 2017*. Springer International Publishing, Cham, 409–437.
- [7] David Bruce Cousins et al. 2017. Designing an FPGA-Accelerated Homomorphic Encryption Co-Processor. *IEEE TETC* 5, 2 (2017), 193–206.
- [8] Craig Gentry et al. 2012. Ring switching in BGV-style homomorphic encryption. In *Proc. ICSCN*. Springer, Springer, Berlin, Heidelberg, 19–37.
- [9] Naina Gupta et al. 2021. PQC Acceleration Using GPUs: FrodoKEM, NewHope, and Kyber. *IEEE TPDS* 32, 3 (2021), 575–586.
- [10] Wonkyung Jung et al. 2021. Accelerating fully homomorphic encryption through architecture-centric analysis and optimization. *IEEE Access* 9 (2021), 98772–98789.
- [11] Chongyang Li et al. 2021. A high speed NTT accelerator for lattice-based cryptography. In *Proc. CISCE*. IEEE, IEEE, Beijing, China, 85–89.
- [12] Ahmet Can Mert et al. 2020. A flexible and scalable NTT hardware: Applications from homomorphically encrypted deep learning to post-quantum cryptography. In *Proc. DATE*. EDA Consortium, San Jose, CA, USA, 346–351.
- [13] Pietro Nannipieri et al. 2021. A RISC-V Post Quantum Cryptography Instruction Set Extension for Number Theoretic Transform to Speed-Up CRYSTALS Algorithms. *IEEE Access* 9 (2021), 150798–150808.
- [14] Deepika Natarajan et al. 2021. SEAL-Embedded: A Homomorphic Encryption Library for the Internet of Things. *IACR CHES* 2021, 3 (2021), 756–779.
- [15] Daniel Petrisco et al. 2020. BlackParrot: An Agile Open-Source RISC-V Multicore for Accelerator SoCs. *IEEE Micro* 40, 4 (2020), 93–102.
- [16] Dayane Reis et al. 2020. Computing-in-Memory for Performance and Energy-Efficient Homomorphic Encryption. *IEEE TVLSI* 28, 11 (2020), 2300–2313.
- [17] M Sadegh Riazzi et al. 2020. HEAX: An Architecture for Computing on Encrypted Data. *Proc. ASPLOS*, New York, NY, USA, 1295–1309.
- [18] Sujoy Sinha Roy et al. 2014. Compact Ring-LWE Cryptoprocessor. In *CHES*. Springer Berlin Heidelberg, Berlin, Heidelberg, 371–391.
- [19] Yang Su et al. 2020. FPGA-based hardware accelerator for leveled ring-LWE fully homomorphic encryption. *IEEE Access* 8 (2020), 168008–168025.
- [20] Sreenath Reddy Thummaluru et al. 2019. Four-Port MIMO Cognitive Radio System for Midband 5G Applications. *IEEE TAP* 67, 8 (2019), 5634–5645.
- [21] Furkan Turan et al. 2020. HEAWS: An Accelerator for Homomorphic Encryption on the Amazon AWS FPGA. *IEEE TC* 69, 8 (2020), 1185–1196.
- [22] Muhammad Arslan Usman et al. 2018. An intrusion oriented heuristic for efficient resource management in end-to-end wireless video surveillance systems. In *Proc. CCNC*. IEEE, Las Vegas, NV, USA, 1–6.
- [23] Guozhu Xin et al. 2020. VPQC: A domain-specific vector processor for post-quantum cryptography based on RISC-V architecture. *IEEE TCAS-I* 67, 8 (2020), 2672–2684.
- [24] Insik Yoon et al. 2019. A 55nm 50nJ/encode 13nJ/decode Homomorphic Encryption Crypto-Engine for IoT Nodes to Enable Secure Computation on Encrypted Data. In *Proc. CICC*. IEEE, Austin, TX, USA, 1–4.