

FHEIns: Fully Homomorphic Encryption Acceleration for Large Data Applications with In-Storage Processing

Xuan Wang*, Tianqi Zhang*, Keming Fan* Augusto Vega[‡], Minxuan Zhou[†], Tajana Rosing*

*University of California, San Diego [†]Illinois Institute of Technology [‡]IBM Research, USA
 {xuw009, tiz014, k4fan, tajana}@ucsd.edu, ajvega@us.ibm.com, mzhou26@iit.edu

Abstract—Recently, the significance of data privacy protection has been growing rapidly. Homomorphic encryption (HE) enables computation directly on ciphertexts, making it attractive for privacy-sensitive databases in cloud datacenters. Although FHE enables privacy-preserving compute, ciphertext expansion and long-latency primitives drive up memory footprint and delay, worsening compute and memory pressure for database search. In practice, encrypted databases span hundreds of gigabytes to terabytes, making the storage I/O the dominant bottleneck. However, most prior FHE accelerators optimize on-chip computation and the main memory traffic while assuming working sets fit in HBM. Therefore, in this work, we present FHEIns, an in-storage processing architecture that executes FHE kernels close to data inside the NAND flash-based solid-state drives (SSDs) to exploit the internal bandwidth of the SSD. FHEIns achieves up to $24.7\times$ and $2.67\times$ speedup compared to the state-of-the-art FHE ASIC accelerators on trending FHE-based database benchmarks.

Index Terms—In-Storage Processing, SSD, Storage, Database, Security, Privacy, Fully Homomorphic Encryption

I. INTRODUCTION

With the continued move to cloud platforms, databases have quickly evolved to become the core infrastructure for routine application workloads [1], [2]. As a result, security and privacy have become rising concerns for large datacenter databases. Fully Homomorphic Encryption (FHE) performs computation in the encrypted domain, allowing cloud processing without plaintext access [3]–[6]. This capability is critical for processing high-volume, sensitive databases in multi-tenant clouds. Across cloud databases and search services, queries decompose into two dominant primitives, **indexing** to narrow candidates and **scoring** to rank them, such as private information retrieval [7], [8], bioinformatics sequence matching [9], [10], and privacy-preserving database searches [1], [2], [11], [12].

Unfortunately, FHE operations are usually $10^4\times$ – $10^5\times$ slower than their plaintext counterparts in existing systems [13]. Recently, prior works proposed a number of dedicated FHE accelerators to bridge this computational gap, such as SHARP [14], ARK [15], Craterlake [13], BTS [16], and UFC [5]. These FHE accelerators utilize dedicated modular arithmetic-based homomorphic encryption (HE) functional units and large on-chip scratchpads (i.e., 256-512MB) with high bandwidth off-chip HBM (i.e., 1TB/s) to develop the very

wide vectorized FHE processing engines for extremely high throughput.

Beyond the computational gap, encrypted databases and auxiliary Galois keys also can inflate the memory footprint by as much as $1000\times$ across the memory hierarchy [5], [13]. Previous FHE accelerators mostly concentrate on the optimization of the on-chip throughput and the contention in main memory traffic, under the implicit assumption that the working set fits within the 64GB HBM memory [5], [13]–[16]. However, for large database applications at scale, raw inputs in the hundreds of GB could even become HE counterparts in the hundreds of TB, invalidating the previous assumptions. The NAND flash-based solid-state drives (SSDs) are required to hold the extensive ciphertext, and therefore shifting the bottleneck to data transferring through the busy PCIe bus with large I/O overhead and inefficiency.

To overcome this I/O bandwidth wall and satisfy some storage-intensive access patterns, in-storage processing (ISP) becomes a promising solution. ISP places computational units inside the storage device to exploit its internal bandwidth and channel-level parallelism, reducing PCIe I/O round-trips and latency. However, applying ISP to the FHE database applications is non-trivial. First, due to the resource constraints for ISPs, the FHE-based database requires a specialized acceleration architecture and dataflow to optimize performance to leverage the internal SSD bandwidth through careful packing strategies. Directly attaching a monolithic FHE accelerator to the flash chips cannot fully leverage the internal parallelism. Second, some FHE operations are severely memory-bound [17]. Heavy load of Galois key (i.e., over hundreds of megabytes), required for each single homomorphic multiplication and rotation, leads into the high on-chip bandwidth requirements through register files and SRAMs for homomorphic functional units.

In this paper, we propose FHEIns, an FHE-based large database accelerator solution with in-storage processing with hardware-software co-design. We introduce a SIMD-aware packing scheme that batches independent elements into aligned vectors to fully exploit the large SSD internal bandwidth rather than the system PCIe bus. In addition, we apply several memory optimization techniques to pin hot keys/ciphertexts in the SSD controller and further reduce

channel contentions between the SSD frontend and backend. Building upon these, we thoroughly explore the characteristics of database computations and design a corresponding hardware ISP architecture, which significantly improves the performance and makes FHE-backed databases practical at scale.

In summary, our contributions are as follows:

- We propose FHEIns, a dedicated HE-based in-storage accelerator architecture. This architecture is optimized for computational patterns in large data applications and fully adapts to their characteristics, significantly enhancing the efficiency of database queries.
- We introduce an ISP-friendly SIMD packing mechanisms for FHE-based database computations to exploit the SSD internal bandwidth.
- We exploit several memory optimization techniques, which aim to reduce on-chip memory requirements by minimizing HE Galois key size and algorithm-hardware co-design.
- We build a cycle-accurate simulator for the FHEIns architecture, which demonstrates up to $24.7\times$ and $2.67\times$ better speedup compared to the state-of-the-art FHE ASIC accelerators on trending FHE-based database benchmarks.

II. BACKGROUND AND MOTIVATION

A. Internal Architecture of Modern SSD

Modern SSDs are based on three main components: the SSD controller, DRAM, and NAND flash packages. An SSD controller comprises (i) several lightweight CPU cores that run the flash firmware/Flash Translation Layer (FTL) and (ii) per-channel flash controllers that handle request sequencing and Error Correction Code (ECC) for the attached NAND. The FTL manages host I/O protocol, address translation, internal scheduling, and device maintenance (e.g., garbage collection, wear-leveling), thereby masking NAND’s erase-before-write and variability from the host [18]. Additionally, modern SSDs employ low-power DRAM (e.g., 4 GB LPDDR4 for a 4 TB drive) for controller metadata. Most of it caches logical-to-physical (L2P) mappings to sustain random-access performance.

SSD capacity is provisioned by dense 3D-NAND flash packages organized hierarchically, channels, chips, planes, blocks, and pages. Contemporary devices commonly expose multiple channels (e.g., 16-32) to harvest internal parallelism across chips and planes [19], [20], where multiple chips share a channel. Within a chip, multiple planes (e.g., 2 or 4) can be accessed at page granularity (e.g., 4-64 KiB in size) via per-plane page buffers. In NAND flash, reads and writes occur at the page level, whereas erase operations occur at the block level.

B. Fully Homomorphic Encryption Basics

In this work, we focus on the CKKS FHE scheme [4] [3] to leverage the high computational throughput of the SIMD schemes. CKKS is a ring learning with error (RLWE) FHE scheme with its secret key sampled from a polynomial ring

\mathcal{R}_Q . Each ciphertext is represented as a pair of polynomial (b, a) in \mathcal{R}_Q , where $\mathcal{R}_Q = \mathcal{R}/Q\mathcal{R}$ and $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$, N represents the ring dimension for the polynomial ring. CKKS packs multiple plaintext elements into a single ciphertext for SIMD processing, which increases the computational throughput and reduces the total memory footprint.

CKKS provides homomorphic multiplication and addition between two ciphertexts or one ciphertext and one plaintext ($HMult$, $PMult$, and $HAdd$). However, homomorphic multiplication between two ciphertexts, $(\mathbf{a}_0, \mathbf{b}_0) \in \mathcal{R}_Q^2$ and $(\mathbf{a}_1, \mathbf{b}_1) \in \mathcal{R}_Q^2$, is complex because the result is a ciphertext $(\mathbf{a}_0\mathbf{a}_1, \mathbf{a}_0\mathbf{b}_1 + \mathbf{a}_1\mathbf{b}_0, \mathbf{b}_0\mathbf{b}_1) \in \mathcal{R}_Q^3$. To transform the result to \mathcal{R}_Q^2 , CKKS requires a key-switching process that multiplies the resulting ciphertext with a special ciphertext, the Galois key switching key (ksk), which could be over hundreds of megabytes (e.g., 360MB as measured in [21]). Additionally, key-switching is also computationally expensive as it requires many (inverse) Number Theoretic Transform ((i)NTT) and modular multiplication with additions. Furthermore, due to the significant noise growth after homomorphic multiplication, a rescaling operation is required to divide the ciphertext by Δ . The ciphertext modulus after rescaling becomes Q/Δ . Therefore, CKKS can only support a limited number of continuous multiplications L . CKKS also supports homomorphic r -rotation ($HRot$) on a single ciphertext (i.e., rotates i -th encrypted element in the packed ciphertext to $(i \cdot 5^r \bmod N)$ -th slot), which is done through automorphism. Similar to homomorphic multiplication, homomorphic rotation is also followed by a key-switching process. As this paper focuses mostly on the application-level discussions and the architectural designs, we refer the readers to [4] [3] for the rest of the detailed discussions on FHE operations and corresponding implementations.

Additionally, we note that FHE programs are completely regular and static since the compute platforms are not allowed to have access to the variable values during the computation. Therefore, branching statements do not exist in FHE programs, which means all required dataflow can be predefined offline.

C. Related Works

There are two kinds of obvious limitations of prior works. On one hand, prior FHE accelerators are based on the assumptions that the working sets will always fit into the HBM main memory [5], [13]–[16]. Therefore, they overlook the heavy-load system PCIe bus data transferring overhead for large-scale practical FHE-based database scenarios. In this regime, wide FHE engines sit underutilized waiting on data, and optimizations that target only compute/DRAM contention systematically underestimate PCIe and SSD costs as noted in Section I.

On the other hand, existing ISP acceleration designs for FHE [7], [8], [10] primarily target narrow, application-specific scenarios with extra small parameters (i.e., $N \leq 2^{14}$ and $L \leq 10$). These choices bound ciphertext size and keep relinearization/rotation key packs modest, simplifying placement and movement inside the drive. However, trending FHE

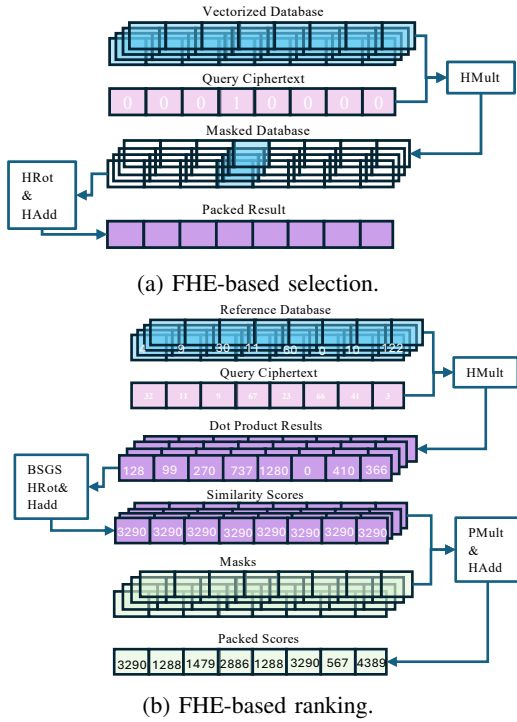


Fig. 1: Examples of FHE-based database compute and storage patterns.

applications are deployed with increasingly large parameter selections (i.e., $N \geq 2^{16}$ and $L \geq 20$) to meet accuracy and security targets. Under these settings, key-switching keys become orders of magnitude larger, stressing controller DRAM, page buffers, and channel bandwidth. Designs tuned to small N and L lack the dataflow machinery needed to feed compute without contenting channels. As a result, these accelerators cannot be easily adapted to larger FHE parameter settings.

III. METHOD

In this section, we present FHEIns, an in-storage processing accelerator architecture for fully homomorphic encrypted database applications. FHEIns aims to accommodate multiple FHE database query algorithms and scales across a range of FHE parameter settings. The result of this section is organized as follows. First, we analyze and characterize FHE-based database applications over their dataflow and SIMD packing strategies. After that, we show the overview architecture of FHEIns SSD. Then, we present the detailed functional units and memory optimization techniques required to support different parameter regimes. Lastly, we present the system support for FHEIns.

A. Compute and Storage Analysis of HE Database

Most encrypted database querying stacks could be decomposed into two categories: **Candidate Selection (Indexing)** and **Candidate Ranking (Scoring)**. Candidate Selection is mostly index-driven, where the goal is to directly retrieve some points or ranges among certain data structures (i.e., list, graph,

etc) with their addresses or keys. Some examples of application scenarios of candidate selection are filtering/aggregations, and tree-structure-based joining/merging. On the other hand, Candidate Ranking is score-based, given that the goal is to sort the database candidates and retrieve certain elements according to the sorting result. For instance, top-k selections, k-nearest-neighbour searches, argmax, and softmax are similar to candidate scoring to produce per-candidate scores, which are commonly used in genomics analysis and machine learning applications. Additionally, privacy-preserving hyperdimensional computing (HDC) [1], [22] also relies on the calculation of cosine similarity scores as its primary backbone for various applications, including but not limited to image/sound/video classification, genomics analysis, and string matching.

Figure 1 shows the detailed algorithmic flow illustration of general **Candidate Selection** and **Candidate Ranking**, respectively. In FHE, selection functions as a generic mask selection over a vectorized database, followed by slot-wise reductions as shown in Figure 1(a). Ranking in FHE is mainly based on similarity score calculation (SSC) as shown in Figure 1(b). $HMult$ is used to get the dot product between the query and the reference database, followed by baby-step-giant-step (BSGS) $HRots$ and $HAdds$ for score summation. And finally, $PMults$ between the ciphertext of the similarity scores and masks with $HAdds$ are utilized to pack similarity scores into a single ciphertext. Under CKKS, all phases are the SIMD operations/layouts followed by a small fixed set of $HRots$ and $HAdds$ for ciphertext packing, which shrinks the memory footprint. As a result, pushing these dataflows inside the SSD exploits the larger internal SSD bandwidth and reduces the data contention on the PCIe.

Given that an enormous number of $HRots$ are involved during the SSC process for summation and packing, Private Information Retrieval (PIR) has a higher $\frac{data}{computation}$ ratio compared to SSC. For the sake of simplicity, we faithfully select PIR to represent **Candidate Selection** and utilize Similarity-Score-Calculation (SSC) as the representation for **Candidate Ranking**. Additionally, in CKKS, for SSC, batching refers to packing multiple elements into a single ciphertext to leverage the computational throughput by applying homomorphic operations on them in parallel. In contrast, since PIR-styled Candidate Selection always requires vectorization, instead of batch within a single ciphertext for reference database, batching means grouping multiple queries and executing them together to leverage data locality.

B. FHEIns Architecture

Figure 2 illustrates the overall architecture and workflow of FHEIns SSD, which integrates the specialized FHE functionality in the commodity SSD controller. The FHEIns SSD controller is mainly composed of the conventional components (i.e., embedded core, flash controller, on-chip network, etc), Channel-level Homomorphic Encryption (HE) accelerators, SSD-level HE Envelope Engine, and with the shared scratchpad and Broadcast Unit (BrU). During the encrypted computation, the client ciphertext metadata will be transferred from

the host system to FHEIns SSD. And the HE constants and databases are loaded sequentially from the NAND Flash chips. The client ciphertext metadata and the HE constants are stored in SSDs' off-chip internal DRAM, shared on-chip embedded DRAM (eDRAM). Additionally, for workloads exceeding the eDRAM size, we leverage the off-chip SSD DRAM to store metadata with extensive memory capacity in order to support the arbitrary parameter sets. The HE constants and databases are loaded sequentially from the Flash DIMs. To leverage the channel-level parallelism and the internal bandwidth, the direct consumption of the original HE data records is performed at the Channel-level HE Accelerator. Further, at the SSD-level HE Accelerator, possible aggregations and intersections are performed to prepare the final results.

C. FHEIns Functional Units

For each HE ACC, the required operations of FHEIns can be realized through a Constant-Geometry Number Theory Transform (CG-NTT) unit, a Base Conversion (BCONV) unit, a shared buffer for Galois keys, and the fully pipelined modular addition and multiplication (MODU) units. FHEIns integrates the CG-NTT [23] for hardware instead of the recursive NTT unit to support large FHE parameters. $\log N$ NTT stages with N butterfly computations are required for an N element polynomial, which becomes the computational performance bottleneck in the recursive NTT. We utilize CG-NTT to support highly vectorized NTT units while reducing costly interconnects and bank-conflicts during (i)NTT memory accesses [5], [23].

Therefore, $HMult$, $PMult$, and $HAdd$ utilize MODUs directly, where $HMult$ is followed by key-switching. In FHEIns, key-switching is performed using modular switching (i.e., modular up and modular down) with the $iNTT \rightarrow$ Base Conversion \rightarrow NTT pattern sequence, which can be done in a fully-pipelined fashion in FHEIns with CG-NTT and BCONV units. We utilize the NTT units with extra NTT and $iNTT$ steps for automorphism in HRot as proposed by [5] to reduce the overall ISP resource overhead.

To balance the utilization of functional units and fulfill the on-chip SRAM requirements for each HE accelerator, an

aggregated design is proposed. Instead of connecting each channel to a separate HE accelerator, FHEIns concatenates K channels to one HE accelerator so that each accelerator receives higher channel bandwidth and large SRAM memory capacity (K is the number of channels whose data are processed by the same HE accelerator). And a separate SSD-level ACC is utilized to support the computations/dataflows that require access to data from all channels.

D. FHEIns Memory Optimizations

We apply two memory optimizations for FHEIns. Firstly, we incorporate some algorithmic-hardware co-design strategies to reduce the parameters required by homomorphic operations. The on-the-fly dynamic twiddle factors generator (OTFGen) dynamically produces twiddle factors required for each stage of (i)NTT to reduce memory requirements. Although twiddle factors for large ring sizes can exceed 10MB (i.e., $N - 1$ twiddle factors), OTFGen needs under 32 KB of seed memory to produce them.

Secondly, we exploit the utilization of the on-chip SSD DRAM to store FHE metadata for FHEIns dataflow. Most on-chip DRAM is occupied by the L2P mapping cache as mentioned in Section II-A. FHEIns reclaims this capacity by persisting the L2P tables to NAND chips and repurposing the GB-class DRAM for storing FHEIns runtime metadata. We carefully design the FHEIns dataflows to only sequentially access the underlying NAND flash chips, so FHEIns requires only a small amount of L2P NAND chip reads, which is an ignorable overhead compared with homomorphic operations. Meanwhile, because FHE programs are branch-free and admit static schedules, all FHEIns metadata is produced offline on the host as a one-time preprocessing step and can be reused across different query batches.

E. System Support

1) *SSD Management Tasks*: Error-correcting codes (ECC) mechanisms are essential in SSDs to detect and correct bit errors caused by NAND cell noise and distortion [24]. FHEIns's ISP accelerators are located in the SSD controller after ECC, so ECC does not restrict FHEIns's ISP performance. Since FHEIns does not require write-backs to the NAND flash chips, it avoids write disturbance errors and reliable retention age degradation [20].

2) *System Integration*: FHEIns can operate in two modes, namely conventional-I/O mode and FHE-acceleration mode. In conventional-I/O mode, FHEIns handles regular storage requests for any privacy-preserving workloads just as a conventional SSD. Note that we did not interfere with the back-end control path of the SSD, allowing FHEIns to the server to efficiently process regular storage requests as normal. The embedded cores will manage the requests and commands. In FHE-acceleration mode, the computational logics are enabled for various FHE operations and commands.

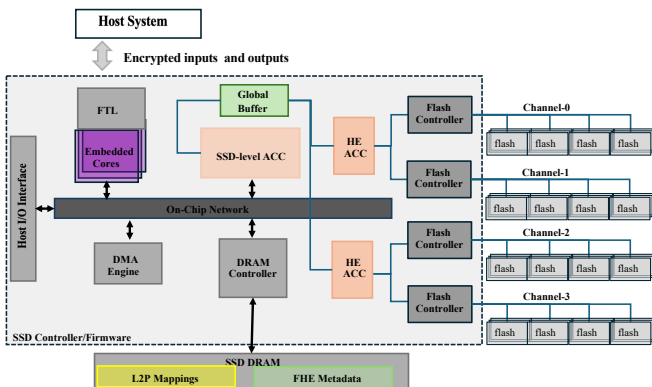


Fig. 2: FHEIns Overview.

TABLE I: SSD Simulation Configurations.

Component	SSD-S Config	SSD-L Config
Controller Host I&F	4 ARM Cortex-A9 cores; NVMe2.0; PCIe 4.0 \times 4	
DRAM	LPDDR5, 51.2GB/s	
DRAM Capacity	4GB	8GB
Flash Memory	16 Channels/SSD 4 Chips/Channel 1600MT/s; 4TiB Capacity	32 Channels/SSD 8 Chips/Channel 2400MT/s; 16TiB Capacity
Shared Flash Parameters	Page size = 32KB; 2 Dies/Chip; 2 Planes/Die; 1024 Blocks/Plane; 1024 Pages/Block; 8-bit Channel Bus;	
Timing Parameters	Read latency(t_R) = 30us; Program time (t_{Prog})=390us; Block Erase Time(t_{BERS})=3800us	

IV. RESULTS

A. Methodology

1) *Modeled Hardware*: To capture the behavior of diverse system configurations, we developed a Python-based event-driven, cycle-level simulator. The design takes inspiration from prior SSD simulation frameworks such as SimpleSSD [25] and MQSim [18], but extends them with fine-grained timing models. Our simulator accounts for the delay of conventional SSD components—including controller processors, on-board DRAM, and multi-level NAND flash—as well as the additional multi-tier processing units incorporated in our architecture. For area and power characterization of our proposed processing units, we describe the designs in Verilog HDL and synthesize them using Synopsys Design Compiler, targeting the open-sourced ASAP 7nm PDK [26]. And we adapted FinCACTI [27] to enable the proper modeling of the interconnects and SRAM components with the published information [28].

2) *SSD Configurations*: To cover a broad spectrum of storage devices, we evaluate FHEIns across two representative SSD classes: a cost-sensitive, small-capacity configuration (**SSD-S**) [19], and a high-performance, large-capacity configuration (**SSD-L**) scaling up according to [20]. Note that both configurations are connected to PCIe Gen4 [29] to match the current SSD market. Taken together, these choices bracket consumer through enterprise deployments and expose variation in interface bandwidth, capacity provisioning, and controller resources, enabling a fair assessment of FHEIns under distinct cost, performance, and I/O regimes.

3) *Workloads*: Without the loss of generality, we evaluate FHEIns over the **Private Information Retrieval (PIR)** and the **Similarity Score Calculation (SSC)** as mentioned in Section III-A. We evaluate FHEIns against the state-of-the-art FHE accelerator baseline, SHARP [14], where we connect SHARP to SSD with PCIe Gen4 [29] and faithfully modeled the performance of SSD data transfer to SHARP. We denote this as our **Baseline** for the following sections. For the sake of fairness, we implement the same PIR protocol as [8]. Specifically, we evaluate FHEIns on PIR primarily using the

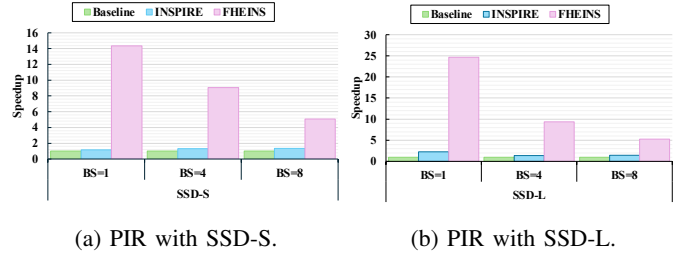


Fig. 3: Overall results for PIR with different SSD configurations with different Batch Sizes (BS).

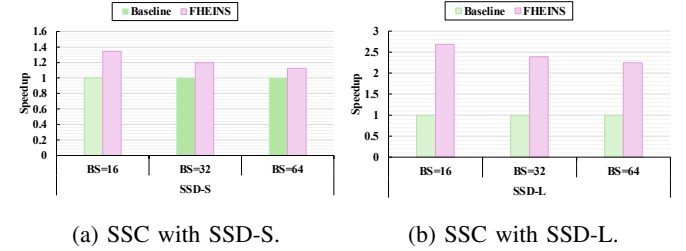


Fig. 4: Overall results for SSC with different SSD configurations with different Batch Sizes (BS). The speedups are normalized to Baselines.

Voice Calling (VCall) dataset as [8]. For FHE parameter selections, we utilize ring dimension $N = 2^{12} = 4096$ and $L = 3$ (i.e., 2 effective multiplication depth) similar to previous works for the **PIR** workloads [7], [8]. For the **SSC** workloads, we utilize $N = 2^{16} = 65536$ and $L = 39$, which satisfies $\lambda = 128$ bits security according to [30]. The dataset is the synthetic dataset similar to [10] with 64 bits batched input queries and 128GB reference database. We note that the concept of batch refers to different meanings for **PIR** and **SSC** as described in Section III-A.

B. Experimental Results

1) *Overall Results*: Figure 3 and Figure 4 demonstrate the overall experimental results for our chosen workloads **PIR** and **SSC**. For the **PIR** workload, FHEIns achieves up to $14.3\times$ and $24.7\times$ speedup for **SSD-S** and **SSD-L**, respectively. FHEIns gains more performance improvement given the enlarged internal SSD bandwidth and richer ISP resources. We observe that FHEIns gains larger speedup on **SSD-L**, which gains from higher aggregated internal SSD bandwidth. We also compare FHEIns with previous in-storage PIR accelerator [8], denoted as **INSPIRE**. Note that **INSPIRE** gains limited performance improvements (i.e., $1.2\times$ to $2.3\times$) compared to **Baseline**. Different from the original paper [8], we utilize the FHE accelerators as the host instead of the CPU, which provides dramatically more on-chip resources and is naturally a stronger baseline. **INSPIRE** has significantly fewer functional units compared to SHARP, so it achieves superior performance.

Figure 4 shows the overall experimental results for our **SSC** workloads, where FHEIns achieves up to $1.3\times$ and $2.7\times$

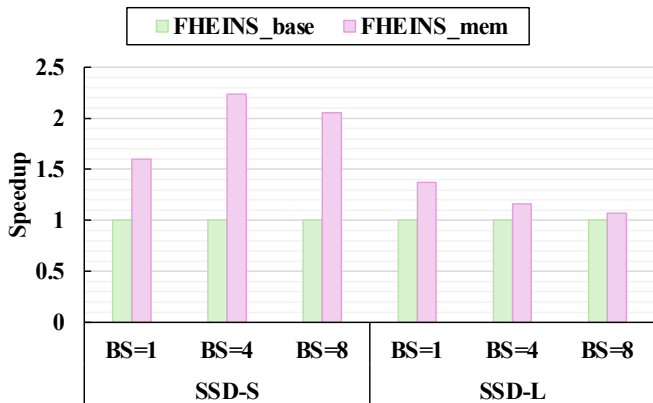


Fig. 5: Ablation study results on memory optimizations.

TABLE II: Area and power specification of FHEIns’s logics.

Logic Unit	Area (mm ²)	Power (mW)
NTT Engine	0.45	786.3
B CONV Engine	0.19	276.3
MODU	0.53	366.1
OT	0.21	296.6
OT Seed memory	0.015	7.4
KeyGen (PRNG)	0.023	9.6
256KB RF	0.005	0.18
7MB SRAM	1.46	653.4
13MB SRAM	2.6	1337.2

speedup for **SSD-S** and **SSD-L**, respectively. We observe that FHEIns gains less performance improvement on the **SSC** workloads compared to the **PIR** workloads. This matches our expectation since **PIR** has a higher $\frac{\text{data}}{\text{computation}}$ ratio compared to **SSC** as we discussed in Section III-A. Different from the **PIR** workloads, we observe that FHEIns gains smaller performance improvements on a large SSD configuration since we did not scale the number of HE_ACC linearly as the aggregated channel bandwidth increases due to the area and power constraints. As a result, the performance of FHEIns is limited by the computational throughput. Note that we are not comparing FHEIns with [10] since [10] is specified for in-flash processing that involves SSD chip-level modifications. Therefore, [10] requires customized algorithms and processing dataflows that are not generalizable to FHEIns, which requires only hardware modifications at the SSD-controller level.

2) *Ablation*: Figure 5 shows the effectiveness of our memory optimization techniques described in Section III-D. **FHEIns_base** represents the hardware setting without any memory optimizations (i.e., all required inputs are retrieved from SSD), and **FHEIns** is our optimized setting with all memory optimizations applied. We observe that FHEIns achieves $1.60\times$ to $2.24\times$ speedup with memory optimizations applied compared to the **FHEIns_base** in Figure 5. We observe that for **SSD-L**, FHEIns gains less from memory optimizations as **SSD-L** provides larger aggregated channel bandwidth for each HE ACC, so that the performance degradation resulting by loading Galios keys from the backend chip decreases.

3) *Area and Power Overview*: Table II presents the area and power overhead of the FHEIns architecture, broken down into the specification of each hardware functional unit component, where for each FU, the overhead is synthesized based on the design for 128 lanes structure. As mentioned in Section III-C, channel-level and SSD-level accelerators each have 7MB and 13MB SRAM, respectively, while each HE accelerator has 4 NTT and B CONV engines paired with 8 MODUs for computational throughput. Note that in the FHEIns architecture, there are 4/8 channel-level accelerators for (**SSD-S**) and (**SSD-L**) respectively. Therefore, the total power overhead is still within the 55W budgets of conventional ISP designs [31] for both cases.

Additionally, we observe that the computational logics, including the NTT Engine, B CONV Engine, and MODU, occupy 40.9%(57.7%) of the area and 65.3%(66.2%) of the power (for configurations corresponding to **SSD-L**), respectively. This shows significantly less memory/buffer occupation compared to previous FHE-ISP accelerators, which is due to our memory hierarchy that offloads part of the FHE metadata to the SSD DRAM. We notice that this breakdown is more similar to previous FHE ASIC accelerators [13].

V. CONCLUSION

To conclude, we present FHEIns, an in-storage processing accelerator architecture for fully homomorphic encrypted database applications. We introduced and analyzed ISP-friendly SIMD packing mechanisms for FHE-based database computations and leverage multiple memory optimization techniques. Putting together our novel hardware architecture and software mechanisms, FHEIns achieves up to $24.7\times$ and $2.67\times$ better speedup compared to the state-of-the-art FHE ASIC accelerators on trending FHE-based database benchmarks.

Our proposed FHEIns accelerator not only improves the performance and energy efficiency of FHE-based database applications but also lays the groundwork for more practical and scalable applications in real-world large-scale scenarios with storage I/O considerations. And to the best of our knowledge, FHEIns is the first attempt to support more than a single FHE workload, which demonstrates the feasibility, portability, and advantages of specialized hardware acceleration for FHE with in-storage processing, positioning it as a viable solution for datacenter-scale, privacy-preserving analytics.

ACKNOWLEDGMENT

This work was supported in part by PRISM and Co-CoSys—centers in JUMP 2.0, an SRC program sponsored by DARPA, and by the NSF under Grants No. 2112665, 2112167, 2052809, and 2211386.

REFERENCES

- [1] Y. Nam, A. Moitra, Y. Venkatesha, X. Yu, G. De Micheli, X. Wang, M. Zhou, A. Vega, P. Panda, and T. Rosing, "Rhychee-fl: Robust and efficient hyperdimensional federated learning with homomorphic encryption," in *2025 Design, Automation & Test in Europe Conference (DATE)*. IEEE, 2025, pp. 1–7.
- [2] P. Antonopoulos *et al.*, "Azure sql database always encrypted," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 1511–1525.
- [3] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2017, pp. 409–437.
- [4] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "Bootstrapping for approximate homomorphic encryption," in *Advances in Cryptology—EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29–May 3, 2018 Proceedings, Part I 37*. Springer, 2018, pp. 360–384.
- [5] M. Zhou, Y. Nam, X. Wang, Y. Lee, C. Wilkerson, R. Kumar, S. Taneja, S. Mathew, R. Cammarota, and T. Rosing, "Ufc: A unified accelerator for fully homomorphic encryption," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2024, pp. 352–365.
- [6] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Computing Surveys (Csur)*, vol. 51, no. 4, pp. 1–35, 2018.
- [7] H. Wang, Y. Yang, J. Zhang, H. Lu, and X. Li, "Ares: High performance near-storage accelerator for the-based private set intersection," in *The 62nd Design Automation Conference (DAC)*, 2025.
- [8] J. Lin, L. Liang, Z. Qu, I. Ahmad, L. Liu, F. Tu, T. Gupta, Y. Ding, and Y. Xie, "Inspire: in-s torage p rivate i nformation re trieval via protocol and architecture co-design," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 102–115.
- [9] X. Wang, M. Zhou, G. De Micheli, Y. Nam, S. Pinge, A. Vega, and T. Rosing, "Pathe: A privacy-preserving database pattern search platform with homomorphic encryption," in *2025 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2025, pp. 1–9.
- [10] M. Kabra, R. Nadig, H. Gupta, R. Bera, M. Frouzakis, V. Arulchelvan, Y. Liang, H. Mao, M. Sadrosadati, and O. Mutlu, "Ciphmatch: Accelerating homomorphic encryption-based string matching via memory-efficient data packing and in-flash processing," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2025, pp. 111–130.
- [11] M. Saraswat and R. Tripathi, "Cloud computing: Comparison and analysis of cloud service providers-aws, microsoft and google," in *2020 9th international conference system modeling and advancement in research trends (SMART)*. IEEE, 2020, pp. 281–285.
- [12] R. J. S. Raj, M. V. Prakash, T. Prince, K. Shankar, V. Varadarajan, and F. Nonyelu, "Web based database security in internet of things using fully homomorphic encryption and discrete bee colony optimization," *Malaysian Journal of Computer Science*, pp. 1–14, 2020.
- [13] N. Samardzic, A. Feldmann, A. Krastev, N. Manohar, N. Genise, S. Devadas, K. Eldefrawy, C. Peikert, and D. Sanchez, "Craterlake: a hardware accelerator for efficient unbounded computation on encrypted data," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 173–187.
- [14] J. Kim *et al.*, "Sharp: A short-word hierarchical accelerator for robust and practical fully homomorphic encryption," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–15.
- [15] —, "Ark: Fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1237–1254.
- [16] S. Kim *et al.*, "Bts: An accelerator for bootstrappable fully homomorphic encryption," in *Proceedings of the 49th annual international symposium on computer architecture*, 2022, pp. 711–725.
- [17] R. Agrawal *et al.*, "Mad: Memory-aware design techniques for accelerating fully homomorphic encryption," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 685–697.
- [18] A. Tavakkol *et al.*, "{MQSim}: A framework for enabling realistic studies of modern {Multi-Queue}{SSD} devices," in *16th USENIX Conference on File and Storage Technologies (FAST 18)*, 2018, pp. 49–66.
- [19] TechPowerUp. (2025) Samsung 990 pro 4 tb ssd specifications. Accessed: 2025-08-19. [Online]. Available: <https://www.techpowerup.com/ssd-specs/samsung-990-pro-4-tb.d863>
- [20] Micron. (2025) Micron 9550 ssd series technical product specification. [Online]. Available: <https://assets.micron.com/adobe/assets/urn:aaid:aem:8c1e2b9b-d81a-45f0-9a9f-8edcd6c23ec9/renditions/original/as/9550-nvme-ssd-tech-prod-spec.pdf>
- [21] N. Neda *et al.*, "Ciflow: Dataflow analysis and optimization of key switching for homomorphic encryption," in *2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2024, pp. 61–72.
- [22] B. Khaleghi, X. Yu, J. Kang, X. Wang, and T. Rosing, "Private and efficient learning with hyperdimensional computing," *IEEE Transactions on Circuits and Systems for Artificial Intelligence*, 2024.
- [23] T. Fritzmann and J. Sepúlveda, "Efficient and flexible low-power ntt for lattice-based cryptography," in *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2019, pp. 141–150.
- [24] K. Zhao *et al.*, "{LDPC-in-SSD}: Making advanced error correction codes work effectively in solid state drives," in *11th USENIX Conference on File and Storage Technologies (FAST 13)*, 2013, pp. 243–256.
- [25] M. Jung *et al.*, "SimpleSSD: Modeling solid state drives for holistic system simulation," *IEEE Computer Architecture Letters*, vol. 17, no. 1, pp. 37–41, 2017.
- [26] L. T. Clark *et al.*, "Asap7: A 7-nm finfet predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.
- [27] A. Shafaei *et al.*, "Fincacti: Architectural analysis and modeling of caches with deeply-scaled finfet devices," in *2014 IEEE Computer Society Annual Symposium on VLSI*. IEEE, 2014, pp. 290–295.
- [28] J. Chang *et al.*, "12.1 a 7nm 256mb sram in high-k metal-gate finfet technology with write-assist circuitry for low-v min applications," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2017, pp. 206–207.
- [29] B. Specification, "Pci express® base specification revision 4.0, version 1.0," 2002.
- [30] B. R. Curtis and R. Player, "On the feasibility and impact of standardising sparse-secret lwe parameter sets for homomorphic encryption," in *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2019, pp. 1–10.
- [31] V. S. Maitlthy *et al.*, "Deepstore: In-storage acceleration for intelligent queries," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 224–238.