Ares: High Performance Near-Storage Accelerator for FHE-based Private Set Intersection

Haoxuan Wang*†§, Yinghao Yang*†, Jinkai Zhang*†, Hang Lu*†§, Xiaowei Li*†§
*SKLP, Institute of Computing Technology, Chinese Academy of Sciences,
†University of Chinese Academy of Sciences,
§Zhongguancun Laboratory, Beijing, China
{wanghaoxuan23p, zhangjinkai22z, luhang, lxw}@ict.ac.cn, yinghao.y@foxmail.com

***Abstract*—Nowadays, the importance of data privacy protection has grown significantly. Privacy Set Intersection (PSI) based on Fully Homomorphic Encryption (FHE) is widely applied in various privacy protection scenarios, such as federated learning and password verification. Nevertheless, the substantial computational demands of FHE and the vast scale of databases in PSI result in inefficient processing, thereby necessitating specialized accelerator architectures to enhance usability. Current general-purpose FHE accelerators do not adequately address the unique requirements of PSI applications, leading to suboptimal data handling and underutilization of hardware, which impedes their effective deployment for PSI acceleration. This paper introduces Ares, a practical hardware-software co-designed FHE-based PSI FPGA accelerator. We propose Lazy Relinearization to optimize redundant calculations in PSI and reduce computational complexity without changing the PSI protocol. At the same time, through the analysis and decoupling of the PSI computing pattern, we design an efficient hardware acceleration architecture that fully utilizes the bandwidth and computing resources of the hardware to achieve excellent acceleration performance. We highlight the following result: (1) a 47.99× speedup relative to CPU; (2) performance improvements of 1.79× and 1.93× over the state-of-the-art FPGA FHE accelerators, Poseidon and FAB, respectively; (3) achieves 7.96× and 10.95× energy efficiency improvement compared to Poseidon and FAB, respectively.**

I. Introduction

In the era of big data, the demand for efficient and secure data processing has become paramount. The concept of privacy computation was proposed to address the issue of data privacy breaches in complex environments. PSI is a foundational cryptographic protocol in privacy computation, allowing two parties to determine the intersection of their sets without revealing any additional information. Among various implementations, the PSI implementation based on FHE is the most widely used. The HE-based PSI protocol can perform set intersections while data remains encrypted. For instance, Microsoft integrated this technology into its Edge browser to detect leakage of user passwords [1]. Similarly, Apple released a homomorphic encryption library in Swift, which it employs in the latest iPhone to support Live Caller ID Lookup functionality [2]. In real-world PSI applications, the database size often easily reaches the level of the billions [3]. Such large computational workloads and databases make it impractical for general-purpose hardware such as CPUs to perform FHE-Based PSI.

Currently, many dedicated FHE accelerators—such as ARK [4], SHARP [5], CL [6], Poseidon [7] and FAB [8] have been developed to accelerate FHE computations. These architectures can also be applied to HE-based PSI. Fig. 1 illustrates the architecture of a traditional FHE accelerator and its main components and processes involved in PSI computation. However, employing these general FHE accelerators to perform PSI queries directly would be extremely inefficient.

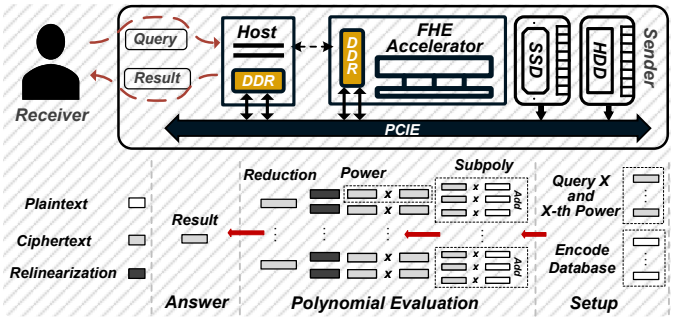

Fig. 1. The data path for executing PSI query with general FHE accelerators. These accelerators are typically connected to the host system as PCIe expansion cards, meaning that when processing large-scale datasets, data must first be read from external storage devices (e.g., SSDs or HDDs).

Firstly, due to the PSI protocol's requirement to perform computations on each entry in the database, large-scale datasets cannot be entirely stored in the onboard memory of FHE accelerators. Instead, they must be stored on the host's SSD or HDD and transferred to the accelerator via the PCIe interface during computation. Although DMA can reduce CPU involvement by facilitating data transfer between the SSD and host, a large portion of execution time is still spent on data transfer over the busy (heavy-load) system PCIe bus.

Secondly, another limitation of using general-purpose FHE accelerators is their low hardware utilization. These architectures are designed to accelerate generic FHE schemes without considering the specific requirements of PSI applications. The PSI query flow, as illustrated in the lower part of Fig. 1, shows that the database is encoded into plaintext polynomial form, where the key computation is polynomial evaluation. This process requires only a subset of FHE computational functions, leading to inefficient use of both computational and bandwidth resources in prior FHE accelerators. Therefore, FHE-based PSI requires specialized hardware acceleration architecture to optimize performance according to its requirements.

Furthermore, the computational pattern in PSI protocols exhibits some redundancy. As shown in the lower part of Fig. 1, the polynomial evaluation involves a relinearization step immediately following each ciphertext multiplication. These two steps are among the most time-consuming in FHE. However, based on the computational pattern of the PSI, relinearization does not need to be performed immediately after every ciphertext multiplication. Therefore, to improve the efficiency of FHE-based PSI, it is necessary not only to conduct targeted designs at the hardware level but also to optimize

Haoxuan Wang and **Yinghao Yang** are co-first authors. **Hang Lu** is the corresponding author.

the computation pattern at the software level.

In this paper, we propose Ares, an FHE-based PSI accelerator solution with hardware-software co-designed based on Near-Data Processing (NDP). We introduce the lazy relinearization optimization strategy to reduce the complexity of polynomial evaluation. Building on this, we thoroughly explore the characteristics of PSI computations and design a corresponding hardware accelerator architecture, which significantly improves the performance of FHE-based PSI queries, making them more suitable for practical applications.

In summary, our contributions are as follows:

- We introduce an optimization termed Lazy Relinearization, which aims to reduce computational complexity by minimizing redundant and costly relinearization operations during the PSI computation process.
- We propose Ares, a dedicated HE-based PSI accelerator architecture. This architecture is optimized for computational patterns and fully adapts to PSI's characteristics, significantly enhancing the efficiency of PSI queries.
- We implemented a prototype of Ares on the commercial NDP platform SmartSSD. The results show that Ares has a 48× acceleration compared to the CPU. Compared to the SOTA FPGA FHE accelerator, it offers a 1.93× improvement in performance and a 10.95× improvement in energy efficiency.

II. Background & Motivation

A. FHE-Based PSI

In a classic FHE-based PSI [9]–[11], the sender holds a larger set X with size $|X|$, and the receiver holds a smaller set Y with size $|Y|$. The receiver encrypts its set $Y = \{y_1, y_2, \ldots, y_{|Y|}\}$ into ciphertexts $Y' = \{c_1, c_2, \ldots, c_{|Y|}\}$. The sender computes an interpolating polynomial $F_X(c) = \prod_{m \in X}(m - c)$ for each c_i, which is the key computation in PSI and also the most complex calculation. At the end of the computation, the receiver decrypts the result: $F_X(c_1), F_X(c_2), \ldots, F_X(c_{|Y|})$, and if the result of any $F_X(c_i)$ is zero, the corresponding y_i is an element of the intersection $X \cap Y$.

The FHE-based PSI protocol is typically implemented using the BFV scheme [12]. In BFV, When calculating $F_{X_1}(c)$, $F_{X_2}(c)$, …, $F_X(c_{|Y|})$, we take $F_{X_1}(c)$ as an example. $F_{X_1}(c) = \sum_{i=0}^{D} a_i c^i$, where D stands for the degree of $F_{X_1}(c)$, a_i is the encoded BFV plaintext of the database, and c represents the encoded BFV ciphertext of the user's set Y. The $F_X(c)$ is called polynomial evaluation.

In polynomial evaluation, both plaintexts and ciphertexts are represented by polynomials. The number of polynomials in a ciphertext or plaintext is denoted as N_p. Initially, N_p is 2 for ciphertexts and 1 for plaintexts. During homomorphic multiplication(Hmul), the N_p of the result is the sum of the N_p values of the two inputs minus one. For instance, if $A = (c_0, c_1)$ and $B = (d_0, d_1)$, then $A \times B = (c_0 d_0, c_0 d_1 + c_1 d_0, c_1 d_1)$. To manage the growth of the polynomial number N_p, relinearization (relin for short) is used to reset the number back to 2.

It can be seen that FHE-based PSI essentially consists of a series of polynomial evaluation computations. Therefore, in this paper, we focus on optimizing and accelerating the polynomial evaluation computations on the sender's side.

B. Redundancy in PSI computation pattern

In the BFV scheme, to prevent the exponential growth of the number of polynomials in ciphertexts during cascaded computations, the relin operation is typically performed immediately after each Hmul. However, in FHE-based PSI, the depth of operation cascades is limited. As shown in Fig. 1, the Hmuls within the polynomial evaluation in FHE-based PSI usually occur at the same depth and do not result in an exponential increase in the number of polynomials. Therefore, performing a relin operation immediately after each ciphertext multiplication is unnecessary in PSI.

Taking the current state-of-the-art (SOTA) APSI protocol [13] as an example, it also fails to recognize this fully and continues to perform a relin after every Hmul. Since Relin is one of the most time-consuming operations in the BFV scheme, this results in significant redundant computation. In Ares, we propose the Lazy Relinearization optimization for this phenomenon, which will be detailed in Sec.III-A.

C. Mismatch with the General FHE Accelerators

There are two obvious limitations to directly using general-purpose FHE accelerators to implement PSI applications.

Low hardware utilization. Traditional accelerator architectures are built to handle large encryption parameters (e.g., $N = 2^{16}$, $\log Q$ over 1500 bits) and complex bootstrapping operations. To support these requirements, they must deploy substantial scratchpad memory (180MB∼512MB) and high-bandwidth off-chip memory (e.g., two HBM modules with 1TB bandwidth) to meet the data R/W demands of the computation units. However, PSI requires significantly smaller encryption parameters (e.g., $N = 2^{13}$, $\log Q$ less than 220 bits), so using these accelerators results in considerable waste of storage and bandwidth resources. Moreover, PSI employs only a subset of FHE operations, excluding functions such as automorphism and bootstrapping, which are commonly supported by general-purpose accelerators. This further reduces hardware efficiency.

Significant data transfer overhead. These accelerators typically face memory limitations when using traditional memory hierarchies, meaning the entire database cannot be fully loaded into on-board memory. However, SSD bandwidth is much lower than that of memory. The commonly used PCIe 3.0×4 interface in SSDs usually has a bandwidth of only about 1.2 GiB/s for this fine-grained ciphertext read operation [14]. During computation, each query requires simple homomorphic operations (i.e., ciphertext addition) with all entries in the dataset (encoded plaintext), and each piece of data is used only once per query. This results in the database being used briefly with no opportunity for reuse. It is difficult to simultaneously transmit the next set of database during computation to mask the significant PCIe communication overhead.

III. Hardware-Software Co-Design for Ares

In this section, we first explore how Lazy Relinearization optimization can effectively reduce the relin overhead in PSI computations, thereby enhancing efficiency. Next, we present design strategies for optimizing the HE computation data flow in PSI, aiming to increase the throughput of PSI computations. Subsequently, we provide a detailed description of the Ares hardware architecture, which is based on these optimizations. Finally, we analyze how the optimized PSI computation workload is efficiently executed on the Ares hardware architecture, ensuring high performance and energy efficiency even when processing large-scale databases.

A. Lazy Relinearization

As discussed in Sec.II-B, there exists a certain amount of redundant computation in the polynomial evaluation. We propose an optimization strategy named Lazy Relinearization (LazyRelin for short), which aims to eliminate these unnecessary computations.

The upper part of Fig. 2 illustrates an example procedure for polynomial evaluation in previous PSI implementations. Here, the

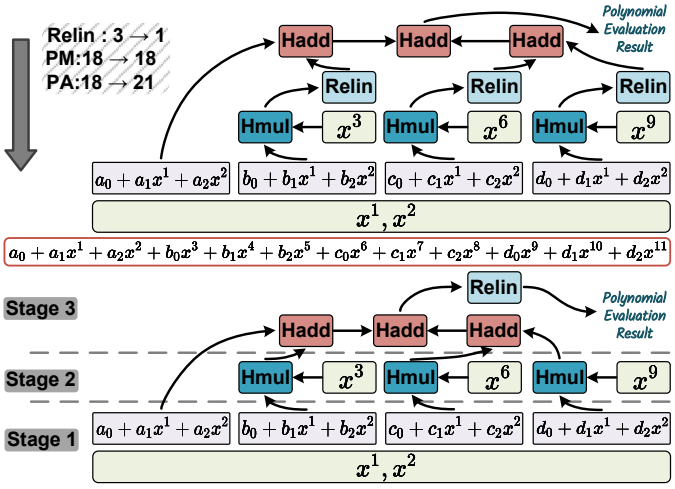

Fig. 2. Example of LazyRelin Optimization. The figure demonstrates the changes in polynomial evaluation computations before and after applying LazyRelin. The red box highlights the polynomial that requires evaluation.

red box denotes a 12th-degree polynomial to be evaluated, where PM stands for polynomial multiplication and PA for polynomial addition. The polynomial is initially decomposed into several sub-polynomials to reduce the required computational power. Notably, in these sub-polynomials, the query x and its powers are in ciphertext form, while other terms are in plaintext. The difference is in Stage ❷, after completing Hmul, we do not perform the relinearization immediately. Instead, we keep the number of polynomials at three until all Hadd aggregations are finished. Then, we perform a single relinearization on the final result.

As mentioned earlier, Hmul operations performed at the same depth do not cause a surge in polynomial numbers. Since all Hmul operations in the polynomial evaluation of PSI occur at the same depth, delaying relinearization is both feasible and merely alters the computation pattern without affecting the accuracy of the PSI protocol execution.

In this example, the polynomial evaluation process eliminates two time-consuming relinearization steps, with the trade-off being a minor increase in PAs, which has negligible overhead. The experimental results of applying this optimization strategy to practical-scale PSI will be presented in Sec.IV-C

B. Architecture

Ares is a NDP architecture that integrates storage, memory, and computing within a single Computational Storage Device (CSD), effectively alleviating bottlenecks associated with data movement over a busy system PCIe bus. The Fig.3 (a) illustrates the overall architecture of Ares. Within the CSD, the programmable circuit contains the computational architecture. And, the remaining components of the CSD consist of DRAM and SSD storage units, along with a PCIe switch. The PCIe switch facilitates communication not only between the host and the device but also supports point-to-point (P2P) communication between the computing circuits and the SSD storage. Databases stored in the SSD are first cached in DRAM before being read into two regions designated for computation, which are partitioned according to the optimized PSI computation pattern. These regions transfer data through a unidirectional Polynomial Evaluation Intermediate Result (PEIR) buffer.

As discussed in Sec.II-A, the primary computational workload in PSI is concentrated in the polynomial evaluation. The database is stored in plaintext form and processed with the query ciphertexts. As shown in the lower half of Fig.2, Stage ❶ involves the computation of sub-polynomials, which includes operations such as plaintext multiplication (Pmul) and ciphertext addition (Hadd). In contrast, Stage ❷ focuses on Hmul, a process with significantly higher computational complexity—typically two orders of magnitude more complex—than the operations in Stage ❶.

Based on these observations, the polynomial evaluation can be decomposed into two components: one is memory-intensive (Pmul and Hadd) and another is compute-intensive (Hmul and relinearization). These two components are computationally independent but exhibit memory locality in their data dependencies. The result of the former is just the input of the latter. However, traditional general-purpose FHE accelerators do not incorporate specialized data pathways tailored to the requirements of polynomial evaluation in PSI, thereby failing to fully leverage the benefits of this decoupling in terms of memory access and computation. Ares could optimize the parallelism and locality inherent between the two components.

As shown in the right half of Fig.3 (a), we divide the computation into two parts in the hardware architecture: the *Memory-Bound (MB) Region* and the *Computation-Bound (CB) Region*. This unidirectional dataflow minimizes redundant data transfer, aligning closely with the characteristics of PSI computations to reduce pressure on memory access. After an assignment of computational resources, these two parts achieved a balance in performance, and a high-performance bubble-free pipeline architecture was implemented with some buffer. The advantages of this unidirectional dataflow will be seen in Sec.III-C. Below we will introduce the Ares computing architecture in detail.

MB Region is responsible for executing Pmul and Hadd. Although these operations have low computational complexity but need to be executed an extremely large number of times, thus requiring high memory bandwidth. Data is retrieved from DRAM and processed through cascaded Pmul and Hadd units to complete sub-polynomial evaluation calculations. The results of these evaluations are temporarily stored in a PEIR buffer before being forwarded to the CB Region for subsequent processing.

CB Region is much more complex. It can be subdivided into two primary components: one module dedicated to Hmul and another unit for the accumulation of multiple sub-polynomials, known as the HAccumulate unit. The hardware design for Hmul includes modular multiplication (MM), modular addition (MA), Number Theoretic Transform (NTT), and a separate Evaluation Key (EVK) buffer. The modular multiplication operation employs Barrett reduction, while the NTT unit adopts an 8-Radix structure inspired by the Poseidon architecture. The HAccumulate unit is a higher-performance variant of the Hadd unit equipped with independent caching. Unlike the Hadd units in the MB Region, the ciphertext polynomials have not undergone relinearization, leading to a larger input dimension for the Hadd computation. The accumulated unrelin results are then transferred via a Local Buffer. Notably, since the basic operators for relin and Hmul computations are identical, coupled with the use of relinearization keys stored in the EVK buffer, the relin process can be completed, with the final output written back to DDR through the outbuffer.

PEIR Buffer can be conceptualized as a dual-port asynchronous FIFO designed to cache the ciphertext results of sub-polynomial computations and transmit them sequentially to the CB Region. Given the generally balanced performance between the two regions, the buffer's capacity does not need to be extensive. In our design, it is configured to hold three ciphertexts.

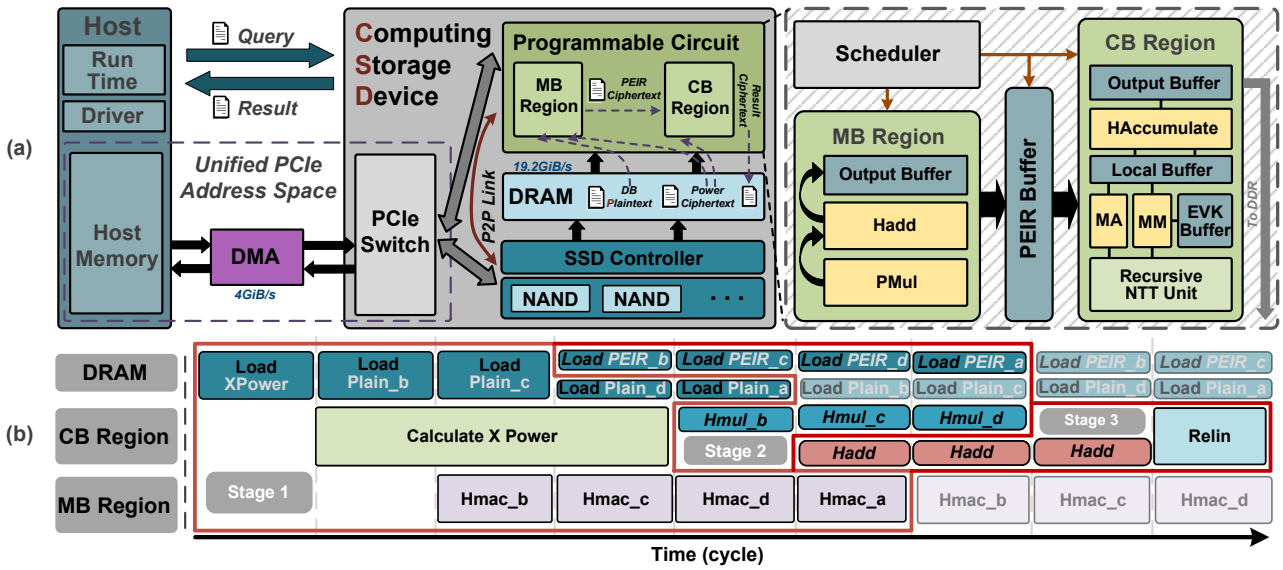

Fig. 3. The left half of part (a) shows the collaborative process between Ares and Host to complete a PSI query, while the right half is the overall computing architecture of Ares. Part (b) illustrates how PSI workload is allocated and executed on two specific regions under the hardware architecture of Ares.

Scheduler is implemented as a finite state machine (FSM) that is responsible for communicating with the Runtime on the Host side. Its primary functions include receiving the ciphertext of the query x and its powers from the host, initiating the transition of the computational unit's state. Upon query completion, the resulting ciphertext is returned to the host.

C. Workload Distribution to different Regions

In Sec.III-A, we explain how the optimized PSI computation pattern performs polynomial evaluation. This section further elaborates on how our hardware design effectively supports these optimizations in conjunction with the Ares hardware architecture. Through a co-design approach that integrates both software and hardware, we have divided the core computational process of polynomial evaluation in PSI into three stages. As shown in Fig.3 (b), we provide a typical polynomial evaluation that illustrates the workload distribution when evaluating a 12th-degree polynomial. This example serves as an extension and concretization of the content described in the lower half of Fig.2.

Stage ❶: At this stage, the plaintext database and the power of x are read from DRAM. The polynomial is divided into several sub-polynomials; for example, *Plain_b* represents b_0, b_1, b_2 in Fig.2. The CB Region calculates the required powers of x, and for the same user, the calculation of x's power needs to be performed only once throughout the process. Meanwhile, the MB Region begins to calculate the sub-polynomials; for instance, *Hmac_b* indicates calculating $b_0 + b_1x^1 + b_2x^2$, and then the result is written into the PEIR buffer.

Stage ❷: This stage focuses on performing efficient Hmul. This part involves multiplying the results of computing sub-polynomials such as *PEIR_b* by x^3, x^6, and x^9 in *Hmul_b*, *Hmul_c*, and *Hmul_d* respectively.

Stage ❸: In the final stage, *Hadd* is utilized to aggregate the results of *Hmul*. Subsequently, only one *Relin* computation needs to be performed on this result to convert the high-order polynomial back into a decryptable form, thereby yielding the complete polynomial evaluation result.

This example remains effective when extended to practical PSI polynomial evaluations. For instance, in the evaluation of a polynomial of degree 2000, during Stage ❶, the degree of the sub-polynomials that need to be computed is 40. For Stage ❷, the required powers of x are x^{40}, x^{80}, etc. Through hardware design, operations such as Hadd, Hmul, and Relin are balanced in performance to ensure that the pipeline can execute efficiently.

There is another subtle optimization worth noting. Observing the order in which plaintext database reads from DRAM, it's evident that the reading of sub-polynomial *plain_a* is not performed at the beginning as one might intuitively expect, but rather delayed until the end of the entire read process. As shown in Fig. 2, after sub-polynomial *plain_a* completes *Hmac_a*, it does not need to participate in the *Hmul* operation in Stage ❷. Therefore, by reading sub-polynomial *plain_a* last, we can directly use it in the *Hadd* computation of Stage ❸ immediately after completing *Hmac_a*, thus better utilizing the limited bandwidth resources. Although the effect of this optimization may not be significant during a single polynomial evaluation. However, when dealing with large-scale databases, where PSI computation involves thousands of polynomial evaluations, this fine-tuning can cumulatively have a positive impact on overall performance.

IV. Evaluation

A. Experimental Setup

Software Platform: We conducted the software performance testing of Ares on a server platform equipped with an Intel Xeon W-1370 CPU, which features 16 threads. The server is also outfitted with 64 GB of DDR4 RAM and 4 TiB of SSD storage.

Hardware Platform: We implement Arse's hardware architecture in SAMSUNG SmartSSD device [15]. The SmartSSD has a 4TB NVMe SSD which directly communicates with Kintex UltraScale+ KU15P FPGA through a PCIe Gen3.0 x4 bus. The attached FPGA has approximately 522K LUTs, 984BRAMs, 1968 DSPs, and DDR4 4GB DRAM.

Baseline: In our experiment, we first evaluate the performance of the basic CPU. Subsequently, we compared Ares with SOTA FPGA-

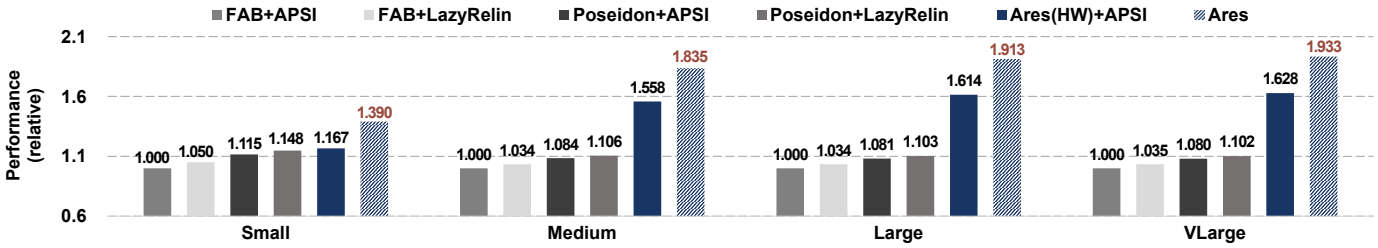

Fig. 4. The ablation experiment demonstrates the performance of LazyRelin optimization on both a general-purpose FHE accelerator and on Ares, across all four benchmarks of different scales.

TABLE I
SIZES OF BENCHMARK DATABASES

Database	Receiver.Num	Sender.Num	DB Size	Entry Length
Small	4096	2^{28}	1.68GB	16B
Medium	4096	2^{30}	6.75GB	16B
Large	4096	2^{32}	27GB	16B
VLarge	4096	2^{34}	108GB	16B

TABLE II
PERFORMANCE COMPARISON OF CPU AND FPGA-BASED FHE ACCELERATORS ACROSS FOUR BENCHMARKS

Platforms		**Small**(s)	**Medium**(s)	**Large**(s)	**VLarge**(s)
CPU		61.82	183.84	711.63	2822.79
FPGA	FAB	1.97	7.19	28.52	113.82
	Poseidon	1.77	6.63	26.38	105.38
	Ares	**1.42**	**3.92**	**14.91**	**58.88**

based FHE accelerators like FAB [8] and Poseidon [7]. None of these accelerators have implemented the PSI protocol, but each accelerator is fully capable of executing PSI. To facilitate comparison, we evaluated each accelerator architecture according to the complexity of the benchmark.

Benchmark: As shown in TableI, we selected four databases of different sizes for evaluation. The size of the receiver (user) set in all four benchmarks is 4096, so the polynomial size N for BFV is configured to 8192. The DB Size refers to the size of the database after undergoing cuckoo hashing and OFPR. For the purpose of experimental evaluation, the specific values in these four databases are randomly generated.

B. Performance

As illustrated in TableII, we initially present an overall system performance analysis for executing identical PSI queries across four databases of varying sizes, using Ares, CPU, and two SOTA FPGA-based FHE accelerators. Ares demonstrates an approximate 47× speedup over the CPU across all four datasets, primarily because HE operations constitute the primary bottleneck for PSI queries under the CPU architecture, rather than memory access becoming the limiting factor. In contrast, for smaller-scale datasets, both Ares and FPGA-based accelerators experience performance enhancements constrained by certain memory access limitations, resulting in Ares achieving only a 1.38× speedup. For larger-scale datasets, specifically Large and VLarge, Ares leverages its advantages in co-designed software and hardware to exhibit nearly double the performance improvement relative to FPGA-based accelerators.

C. Ablation

Due to the excessive resource consumption of FPGA Baseline FAB and Poseidon, they cannot be deployed on the resource-limited CSD platform. We evaluated the performance improvement of deploying LazyRelin optimization on these architectures based on their design complexity. In Figure 4, we normalized the data to the performance of FAB+APSI. The results show that, across four different benchmarks, the performance improvement of the two baselines with LazyRelin optimization was only approximately 3%, exhibiting a similar minor enhancement trend in each benchmark. This limited improvement is attributed to the general FHE accelerator architecture's inability to effectively align with the specific computational patterns of PSI.

In contrast, even using the original APSI computation patterns, the Ares architecture still performs better than the general FHE accelerator. Moreover, after applying the LazyRelin optimization, Ares's performance was further significantly improved, especially in the VLarge database tests, where the maximum performance achieved up to 1.933×. These results not only highlight the high compatibility between the LazyRelin optimization and the Ares architecture but also indirectly confirm the high alignment of the Ares architecture with the PSI computation patterns in its design.

D. Resources And Utilization

Ares exhibits significant efficiency in resource utilization. As shown in the left half of TableIII, a comparison is made between Ares and two other FPGA-based FHE accelerators regarding their primary resource consumption. The data indicate that Ares achieves a reduction of 2∼5× compared to previous FPGA accelerators, whether it pertains to the usage of logical or storage resources. The right half of TableIII evaluates the pure computational performance of the core calculation in PSI query—polynomial evaluation, where D represents the degree of the polynomial. The findings reveal that, despite a marked reduction in hardware resource usage, Ares performs slightly lower than Poseidon in polynomial evaluation but is comparable to FAB. This phenomenon can be attributed to the fact that traditional FPGA-based FHE accelerator designs have not adequately considered the characteristics of PSI computations, lacking dedicated data paths optimized for PSI tasks. Consequently, even with substantial investments in computational and storage resources, the actual utilization rate of hardware remains low. In contrast, Ares effectively maximizes the performance of PSI computations through the optimal use of limited hardware resources, demonstrating its advanced and targeted design approach.

TABLE III
RESOURCE UTILIZATION ANALYSIS: COMPARISON OF RESOURCE USAGE BETWEEN ARES AND BASELINE FPGA ACCELERATORS; POLYNOMIAL EVALUATION LATENCY

Works	Resources				Latency(ms)	
	LUT(K)	FF	BRAM	DSP	D=2^{14}	D=2^{18}
FAB	899	2,073	3,840	5,120	94.4	1511.3
Poseidon	728	977	2,048	8,640	61.3	981.8
Ares	**343**	**474**	**912**	**1,728**	80.5	1288.7

TABLE IV
EFFICIENCY ANALYSIS. ENERGY-DELAY PRODUCT (EDP) IS USED AS THE METRIC, WHERE LOWER IS BETTER. MAX GAIN INDICATES THE MAXIMUM GAIN ACROSS FOUR BENCHMARKS.

Works	Small	Medium	Large	Vlarge	Max Gain
FAB	464.15	6,182	97,281	1,549,417	**10.95**
Poseidon	317.67	4,457	70,564	1,126,041	**7.96**
Ares	**82.26**	**626**	**9,070**	**141,447**	**1.0**

E. Energy Efficiency

To evaluate Ares' performance in terms of energy efficiency, we adopted the Energy-Delay Product (EDP) as the metric. The power consumption data for FAB and Poseidon were obtained through the Xilinx Power Estimator (XPE) [16], while Ares' power consumption data were derived from Power Reports generated after synthesis using the Vitis Development Tool (VDT). It is worth noting that the EDP data for the CPU are not listed in the table because its EDP value is significantly higher—approximately 5500× greater than the accelerators, hence not directly represented in the table. As shown in TableIV, Ares demonstrates superior energy efficiency across four benchmarks, achieving a 10.95× increase in energy efficiency compared to FAB and a maximum 7.96× improvement over Poseidon. On one hand, these notable gains in energy efficiency can be attributed to Ares' efficient utilization of hardware resources, which allows it to maintain high-performance PSI computation capabilities even at lower power levels. On the other hand, the gains are also due to its architecture specifically designed for PSI computations and optimized data transfer paths, further enhancing Ares' competitive edge in energy efficiency.

V. CONCLUSION

In this paper, we present Ares, a hardware-software co-designed accelerator for FHE-based PSI. We optimize the computation pattern using the LazyRelin strategy, which significantly reduces the number of relinearization operations, one of the most time-consuming operations in FHE computations. In terms of hardware architecture design, Ares leverages NDP to minimize data transfer latency and designs matching hardware computational circuits tailored to the PSI computation pattern, thereby maximizing the accelerator's performance.

The proposed Ares accelerator not only enhances the performance and energy efficiency of FHE-based PSI but also paves the way for more practical and scalable applications in real-world scenarios. Overall, our work demonstrates the feasibility and advantages of specialized hardware acceleration for FHE-based PSI, making it a promising solution for large-scale, privacy-preserving data processing. We hope that the progress made in this research will contribute to the broader field of privacy-preserving computation.

VI. ACKNOWLEDGMENT

This work was supported in part by the National Natural Science Foundation of China under Grant 62172387; in part by the CCF-Phytium Fund 2023.

REFERENCES

[1] "Password monitor: Safeguarding passwords in microsoft edge," https://www.microsoft.com/en-us/research/blog/password-monitor-safeguarding-passwords-in-microsoft-edge/, Jan. 2021, microsoft Research.

[2] "Understanding how live caller id lookup preserves privacy," https://docs.developer.apple.com/documentation/sms_and_call_reporting/understanding_how_live_caller_id_lookup_preserves_privacy, apple.

[3] D. Kales, C. Rechberger, T. Schneider, M. Senker, and C. Weinert, "Mobile private contact discovery at scale," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1447–1464.

[4] J. Kim, G. Lee, S. Kim, G. Sohn, M. Rhu, J. Kim, and J. H. Ahn, "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.

[5] J. Kim, S. Kim, J. Choi, J. Park, D. Kim, and J. H. Ahn, "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.

[6] 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.

[7] Y. Yang, H. Zhang, S. Fan, H. Lu, M. Zhang, and X. Li, "Poseidon: Practical homomorphic encryption accelerator," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 870–881.

[8] R. Agrawal, L. de Castro, G. Yang, C. Juvekar, R. Yazicigil, A. Chandrakasan, V. Vaikuntanathan, and A. Joshi, "Fab: An fpga-based accelerator for bootstrappable fully homomorphic encryption," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 882–895.

[9] H. Chen, K. Laine, and P. Rindal, "Fast private set intersection from homomorphic encryption," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1243–1255.

[10] H. Chen, Z. Huang, K. Laine, and P. Rindal, "Labeled psi from fully homomorphic encryption with malicious security," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1223–1237.

[11] B. Pinkas, T. Schneider, G. Segev, and M. Zohner, "Phasing: Private set intersection using permutation-based hashing," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 515–530.

[12] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptology ePrint Archive*, 2012.

[13] K. Cong, R. C. Moreno, M. B. da Gama, W. Dai, I. Iliashenko, K. Laine, and M. Rosenberg, "Labeled psi from homomorphic encryption with reduced computation and communication," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 1135–1150.

[14] L. Daoud, G. Sun, and H. Huen, "Peer-to-peer data transfer evaluation in smartssd-based multi-devices system," in *Proceedings of 36th International Conference on Computer Applications in Industry and Engineering*, vol. 97, 2024, pp. 72–80.

[15] P. Mehra, "Samsung smartssd: Accelerating data-rich applications," *Flash Memory Summit*, 2019.

[16] "AMD Power Estimator." [Online]. Available: https://www.amd.com/en/products/adaptive-socs-and-fpgas/technologies/power-efficiency/power-estimator.html