Trident: the Acceleration Architecture for High-Performance Private Set Intersection

Jinkai Zhang, Yinghao Yang, Zhe Zhou, Zhicheng Hu, Xin Zhao, Liang Chang, Hang Lu, Xiaowei Li, *Senior Member, IEEE*

***Abstract*—Private Set Intersection (PSI) is imperative in discovering the properties of the same data owned by two competitive parties, without revealing anything else of their respective data asset. Existing PSI solutions such as APSI and ORI-PSI suffer from severe communication and computation overhead due to inefficient communication and FHE polynomial evaluation, which hinders their deployment in practice. This issue is evident in both the upper-level protocol and the lower-level hardware platform. In this paper, we propose a novel software/hardware co-design acceleration architecture for PSI, termed as "Trident", which includes two tightly coupled segments: from the protocol perspective, we investigate existing bottlenecks and propose a new PSI protocol with significantly less communication and computation under the security guarantee; besides, we re-architect the hardware platform by designing a PSI-specific accelerator, implemented with both FPGA and ASIC, targeting the key operations in the proposed protocol. We build a real-world experimental environment with two instantiated parties to verify the acceleration architecture, and highlight the following results: (1) up to 130×/145× speedup for the computation of *receiver* and *sender* parties; (2) up to 37× reduction of communication overhead. (3) up to 93,651× and 74,326× higher energy efficiency over the CPU-based ORI-PSI and APSI, respectively.**

***Index Terms*—Private set intersection (PSI), fully homomorphic encryption (FHE), FPGA accelerator, privacy computing**

I. Introduction

PRIVATE Set Intersection (PSI hereafter) is a privacy-preserving technique that enables two mutually untrusted parties to compute the intersection of their data assets, without giving up on their individual privacy. The result is the data that both parties share. The PSI concept is illustrated in Figure 1. Two parties apiece own the private data asset about the customer personal information, some of which overlap with each other. However, they focus on different attributes of the same customer (e.g., Alice in the figure). Usually, the party with fewer data assets that issues the PSI is regarded as the "*receiver*", e.g., party B, while the party with much more data assets that responds to the PSI request from the receiver is regarded as the "*sender*", e.g., party A. The set intersection could be used for the *receiver* to extend its business, e.g., providing more personalized medical service to Alice. PSI has been very widely used in privacy-critical scenarios and industry [1], [4], [12], [26], [30]; for example, Google utilizes PSI as a Chrome extension to check if a username and password entered on a website has been compromised [26]. Alibaba Inc. utilizes PSI in its "datatrust" platform providing multi-lateral data analytics and training to startups [1].

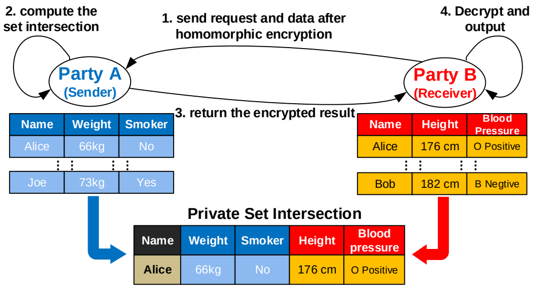


Fig. 1. General concept of Private Set Intersection (PSI).

The first-order principle of PSI is the guarantee of privacy and the privacy is twofold. Party A (the *sender*) must keep its exclusive data (like Joe in the figure) secret, while party B (the *receiver*) needs to encrypt all the data that are transferred to the party A for computing the set intersection. Thus, two parties perform the PSI without learning the exact data under computation, which preserves the privacy.

The limitations of prior PSI. *In general, PSI is very computation-and-communication intensive.* The most state-of-the-art PSI protocol - APSI [10] requires 53 hours of computation and 578GB of data communication for the two parties to complete the PSI. The huge overhead hinders PSI in realistic industrial deployment. The reason is that PSI usually handles huge amount of data asset - from tens of millions to even hundreds of billions. To make things worse, the data must be encrypted to preserve the *receiver*'s privacy that further exacerbates the communication and computation burden.

Existing PSI protocols consistently abide by the principle of encrypting all the data assets before transferring out of the domain. This "complete-data-encryption" paradigm is undoubtedly able to guarantee the privacy of the data owner; the performance of the PSI however, is compelled to the pressure of ciphertext handling. In specific, the existing PSI solutions can be categorized as Table I shows, according to the underlying cryptography - Oblivious Pseudo-random Function (OPRF hereafter) and Fully Homomorphic Encryption (FHE hereafter). These PSI protocols are secure against semi-honest *sender* and malicious *receiver*. Many works like [15] and [20] utilize OPRF as a central primitive in PSI to strengthen the security against malicious *receiver*. Cryptographic primitives in these works include ECC [20], LOWMC-GC [20], etc. OPRF requires computing the encryption parameters and the

Jinkai Zhang and Yinghao Yang contributed equally to this work. Corresponding authors are Jinkai Zhang and Hang Lu. This work was supported in part by the National Natural Science Foundation of China under Grant 62172387; in part by the Youth Innovation Promotion Association of Chinese Academy of Sciences (CAS) under Grant 2021098; in part by the Open Research Fund of the State Key Laboratory of Blockchain and Data Security, Zhejiang University.

TABLE I
EXISTING PSI SOLUTIONS AND OURS. THE DATA ASSET FOR COMPUTING PSI IS 10 BILLION AND 100 MILLION 64-BIT ID NUMBERS RELEASED BY THE INDUSTRY FOR THE *sender* AND *receiver* RESPECTIVELY. THE EXPERIMENTAL SETUP IS CONSISTENT WITH SECTION V.

PSI Solutions	Feature	Platform	Datascope	Cryptography	Comp.	Comm.	Overall Time
LOWMC-GC-PSI [20]	Protocol Only	CPU	$(0, 2^{64})$	OPRF, LowMC-GC	136,521s	3,072GB	388,196s
ECC-NR-PSI [20]	Protocol Only	CPU	$(0, 2^{64})$	OPRF, NR-ECC	143,796s	775GB	207,351s
ORI-PSI [6]	Protocol Only	CPU	$(0, 2^{32})$	OPRF, FHE	159,744s	614GB	210,003s
APSI [10]	Protocol Only	CPU	$(0, 2^{32})$	OPRF, FHE	144,077s	578GB	191,460s
Trident **(ours)**	Protocol & Hardware	Accelerator	$(0, 2^{32})$	OPRF, **PoM-FHE**	**FPGA: 735s** **ASIC: 221s**	17GB	**FPGA: 2127** **ASIC: 1613s**
APSI [10]	Protocol Only	CPU	arbitrary	OPRF, FHE	555,458s	2290GB	743,054s
MT-PSI [31]	Protocol Only	CPU	arbitrary	OPRF, FHE	555,458s	578GB	602,808s
Trident **(ours)**	Protocol & Hardware	Accelerator	arbitrary	OPRF, **PoM-FHE**	**FPGA: 2800s** **ASIC: 842s**	42.4GB	**FPGA: 6273s** **ASIC: 4315s**

negotiation between the two parties, which is one of the most time-consuming and communication-intensive procedures in PSI. For instance, LOWMC-GC-PSI [20] incurs more than 3 TB transmission between the two parties. Huge communication induces a significantly increased overall PSI time (108 hours). This limitation motivates the community to explore the more advanced PSI solution to optimize the communication overhead. FHE is regarded as an efficient way to alleviate this problem. It batches the data during encryption so the ciphertext of the *sender* are much less in volume than the non-FHE based cryptography. By introducing FHE in PSI protocol, [6] reduces the total communication to 614 GB. On top of [6], [10] further leverages OPRF, enables arbitrary datascope and the communication is reduced to 578 GB. MT-PSI [31] proposes the virtual Bloom filter (VBF) and polynomial links (PoL) techniques for long item situation, where VBF encodes long item into several independent shorter items and PoL slices long item into shorter pieces and build links between them. Within the APSI framework, it uses a hybrid scheme of VBF and PoL to optimize the balance between *sender* offline preprocessing time and total communication overhead. In the long item situation, MT-PSI achieves up to a 1712 GB reduction in communication overhead compared to APSI. These FHE-based PSI require the *receiver* and the *sender* to encrypt and transmit a large number of FHE ciphertexts, resulting in a total communication overhead that can reach several hundred GB. In these FHE-based PSI protocols, both computation and communication overhead are performance bottleneck.

Our contribution. Considering the two-sided overhead, the desirable PSI solution should be designed with less communication as its first-order design goal, and simultaneously avoid the computation bottleneck caused by the key techniques like FHE and OPRF. This requires a software/hardware co-design approach for PSI. However, existing solutions all focus on the software level, that is, trying to minimize the complexity on the protocol level to decrease the communications. Despite the insufficient communication reduction, none of them has ever considered re-architecting the existing hardware platform to collaborate with the optimized PSI protocol, which makes the deployment of PSI very difficult; For example, it takes 58 hours for Origin-PSI (ORI-PSI hereafter), 53 hours for APSI, 57.5 hours for the non-FHE based ECC-PSI to compute the intersection only once on commercial off-the-shelf servers. Therefore in this paper, we propose a software/hardware co-design acceleration architecture, termed as "*Trident*", to achieve efficient PSI. Trident involves two tightly coupled segments:

(1) A novel PSI protocol, termed as "*Trident-PSI*". As the topmost novelty, we use "ciphertext digest" under the privacy guarantee, instead of the "full ciphertext" manner widely adopted in existing PSIs. "Full ciphertext" encrypts complete data while "ciphertext digest" encrypts some data completely and uses cryptographic hash to generate digest for the other data. Due to this substitution, we propose Differential Sharing, Trident Initialization and Trident Evaluation which reduce the contribution of the ciphertext transmission to the communication. Moreover, our protocol decreases the FHE ciphertext multiplications, replaced by the plaintext NTT (Number Theoretic Transform) operations on the polynomials. We call this step as Trident PoM (Trident Polynomial Modification). Therefore, Trident-PSI alleviates the communication and computation in tandem on both parties.

(2) A practical PSI accelerator, termed as "*Trident-accelerator*". It targets the specific computation bottleneck introduced by PoM-FHE and OPRF. Based on commercial FPGA, we implement all the hardware acceleration modules needed by the Trident-PSI protocol. We also implement the Trident accelerator ASIC using 7nm technology node.

We highlight the following results: (1) Protocol performance: communication in total is reduced from 614 GB to 17 GB (21.6× and 43.2× for the *receiver* and *sender* party, respectively); (2) FPGA Accelerator performance: it outperforms CPU by 130× and 145× on both parties, and achieves 5× to 10× performance improvement than other FPGA-based accelerators; (3) Overall PSI performance: up to 98.7× speedup via 100M Ethernet connection; (4) Efficiency: the *sender* and *receiver* party is improved by 93,651× and 1,102,960×, respectively; (5) Software/Hardware ablation study: we accumulatively verify the key steps in Trident-PSI to prove its efficacy; (6) ASIC performance: the Trident accelerator ASIC achieves 433× and 486× speedup on the *receiver* and *sender* compared with CPU.

II. BACKGROUND AND MOTIVATION

In this Section, we introduce the basic framework of PSI with FHE (Section II-A), analyze the communication and computation bottlenecks in PSI (Section II-B), and summarize the acceleration opportunities for PSI (Section II-C). Abbreviations and explanations are summarized in Table II.

A. PSI Formulation

In classic two-party PSI, the *sender* holds the data asset X of size N_X, while the *receiver* holds the data asset Y

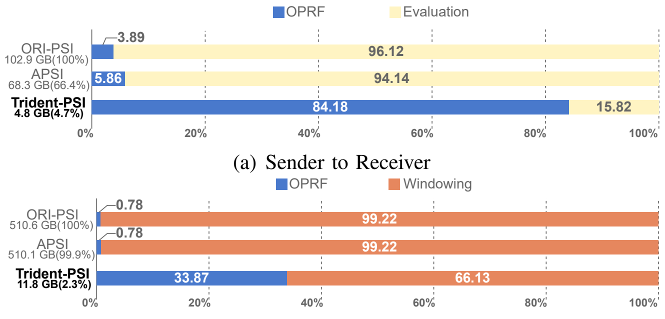
TABLE II
ABBREVIATIONS AND EXPLANATIONS.

Params	Description
Protocol	
NTT	Number Theoretic Transform.
RNS	Resident Number System.
OPRF	Oblivious Pseudo random Function.
ORI-PSI	Origin PSI [6].
PoM	Polynomial Modification.
DPD	Data Partitioning and Differential Sharing.
TIE	Trident Initialization, PoM and Evaluation.
Hardware	
BRAM	Block Random Access Memory.
DRAM	Dynamic Random Access Memory.
SRAM	Static Random Access Memory.
DDR	Double Data Rate DRAM.
HBM	High-Bandwidth Memory.
PCIe	Peripheral Component Interconnect Express.
XDMA	Xilinx Direct Memory Access.
MA/MM	Mod Add/Multiplication.
SM	Shared mod.
HMult/HAdd	Homomorphic Multiplication/Addition.
EC	Elliptic Curves.
ECMult	scalar multiplication on elliptic curves.
LD	Lopez&Dahab.
FSM	Finite state machine.
BMult	Binary finite field multiplication.
BSqr	Binary finite field square.
BModInv	Binary finite field modular inverse.
EDP	Energy Delay Product.
DSP	Digit Signal Process.
RTL	Register Transfer Level.

of size N_Y. According to [10], data asset in X and Y are assumed to be uniformly distributed random variables. After the *receiver* encrypts $Y = \{y_1, y_2, \ldots, y_{N_Y}\}$ as $Y' = \{c_1, c_2, \ldots, c_{N_Y}\}$ (possibly using FHE), the *sender* generates a non-zero random number r_i for each $c_i \in Y'$ and computes $d_i = r_i \prod_{x \in X} (x - c_i)$. By iterating over $x \in X$, the resulting d_i constitutes an encrypted PSI result indicating whether the plaintext in c_i from the *receiver* matches any element of the *sender*'s data asset X. After transferring the ciphertext d_1, d_2,..., d_{N_y} back to the *receiver*, it decrypts each d_i for the plaintext. If the plaintext of some d_i equals to 0, y_i will appear in $X \cap Y$. Encrypting the *receiver*'s data asset using FHE not only provides security against the *sender* but also enables arithmetic operations on ciphertexts for computing the intersection polynomial and obtaining the PSI result d.

OPRF. To further protect the sender's data asset from malicious *receiver*, OPRF must be employed [13], [15], [20]. [6], [10], [31] employ Diffie-Hellman-based OPRF in PSI. It enables the *receiver* to get $S_Y = \{H(y)^\beta : y \in Y\}$ secretly and the *sender* to get $S_X = \{H(x)^\beta : x \in X\}$, where β is secret key of the *sender* and $H(.)$ is a random oracle hash. After OPRF, S_X and S_Y are encrypted under β. According to [6], from the *receiver* view, the distribution of the *sender*'s data after OPRF is indistinguishable from a uniformly random distribution. Thus, OPRF protect the *sender* from malicious *receiver*.

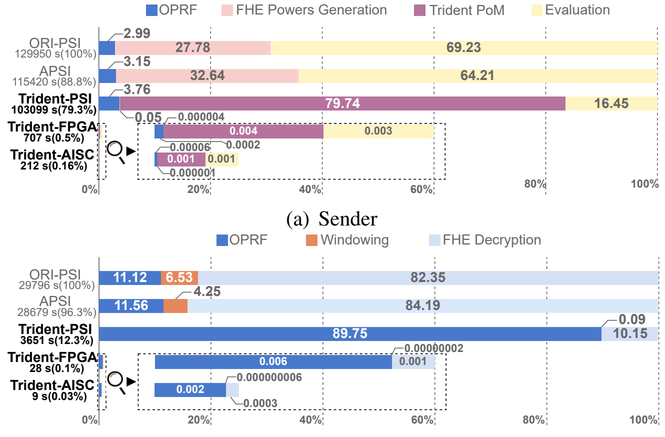
In order for the receiver to get S_Y secretly, the receiver needs to generate a key α and encrypt their dataset to get $Y' = \{H(y)^\alpha : y \in Y\}$. After sending Y' to the sender, the sender re-encrypts Y' using β to get $Y'' = \{(H(y)^\alpha)^\beta : y \in Y\}$ and sends it back to the receiver. Finally, the receiver decrypts it to obtain $S''_Y = \{(H(y)^{\alpha\beta})^{1/\alpha} : y \in Y\} = \{H(y)^\beta : y \in Y\}$.



(a) Sender to Receiver

(b) Receiver to Sender

Fig. 2. Communication breakdown of PSI protocols.



(a) Sender

(b) Receiver

Fig. 3. Time breakdown of PSI protocols. ORI-PSI, APSI and Trident-PSI are evaluated on CPU. The data of Trident-FPGA and Trident-ASIC are normalized to Trident-PSI.

Trident also employs Diffie-Hellman-based OPRF. Section III-B describes more details about the OPRF in Trident.

B. Bottlenecks in Computation and Communication

The difficulty of PSI deployment stems from the intrinsic bottleneck in existing protocols and hardware platform employed. As two baseline PSI representatives, we profile the characteristics of the PSI protocol proposed in [6], [10], referred to as ORI-PSI and APSI respectively, from two aspects:

Communication. Huge communication forms the bottleneck in PSI and must be optimized according to Section I. We establish the *sender* and *receiver* party on our real-world experimental platform (see Section V), and profile the communication actions during PSI. As shown in Figure 2, the communication of the data assets after FHE may attain even 431.67 GB from the *sender* to the *receiver* and 22.55 GB from the *receiver* to the *sender* for APSI. "Windowing" serves to compute the "powers" of the *receiver*'s data needed by FHE and send these powers to the *sender* in the ciphertext format. The two baselines all follow "full ciphertext", so this step exhibits significant communications. "Evaluation" issues the actual intersection computation, in which it segments the high-degree polynomial into multiple low-degree polynomials, performs PSI and then sends the result back to the *receiver*. The result is still in ciphertext that leads to high communication.

Computation. PSI is computationally intensive. All steps of ORI-PSI [6] and APSI [10] involve certain computations to fulfill the whole protocol. However, the most significant computation is also introduced by FHE. Figure 3 clearly proves that the computation time is predominated by the steps that entail FHE - around 97% in total. There are two factors that

Input: Receiver inputs set Y ⊂ $\{0, 1\}^\sigma$ of size N_Y; sender inputs set X ⊂ $\{0, 1\}^\sigma$ of size N_X. N_X, N_Y, σ are public.
Output: Receiver outputs X ∩ Y.

Offline	**Setup**		*Receiver* and *Sender* agree on homomorphic encryption scheme, elliptic curve with order q. *Receiver*: generates parameters of FHE, and samples private input $\alpha \leftarrow \mathbb{Z}_q^*$. *Sender*: samples private input $\beta \leftarrow \mathbb{Z}_q^*$. Set cuckoo table size m and θ=3 random hash functions.
	Sender	**Receiver**	
	① Data Partitioning		*Sender*: Divides data asset into $X_1, X_2 \ldots, X_\Gamma$, then publish lower bound and upper bound of each subset. *Receiver*: Divides data asset into $Y_1, Y_2 \ldots, Y_\Gamma$ according to boundary information according to *Sender.*
	② OPRF Initialization		*Sender*: For each subset X_i, generates $S_{X_i} = \{H(x)^\beta : x \in X_i\}$. *Receiver*: For each subset Y_i, generates $Y_i{}' = \{H(y)^\alpha : y \in Y_i\}$. H(.) is the random oracle hash function.
	③ Multi-hashing		For each S_{X_i}, builds up multi-hash table $\boldsymbol{\mathcal{B}}_i$ with m rows, each has n columns.
	④ Trident Initialization		For each two multi-hash table $\boldsymbol{\mathcal{B}}_{2j\text{-}1}$ and $\boldsymbol{\mathcal{B}}_{2j}$, Splits $\boldsymbol{\mathcal{B}}_{2j\text{-}1}$ and $\boldsymbol{\mathcal{B}}_{2j}$ vertically into δ parts. For each row v of each part i, and for each $\Delta \in Z_p$, computes $F_{i,j,v,\Delta}(x)$ and $D_{i,j,v,\Delta}$.
Online	**Sender**	**Receiver**	**During the j-th iteration of the online step, intersection is computed on X_{2j}, Y_{2j} and X_{2j+1}, Y_{2j+1}, concurrently.**
	❶ OPRF Online		Sends $Y_{2j}{}'$ and $Y_{2j+1}{}'$, receives $Y_{2j}{}'' = \{(y')^\beta : y' \in Y_{2j}{}'\}$ and $Y_{2j+1}{}'' = \{(y')^\beta : y' \in Y_{2j+1}{}'\}$. Then generates $S_{Y_{2j}} = \{(y'')^{1/\alpha} : y'' \in Y_{2j}{}''\}$ and $S_{Y_{2j+1}} = \{(y'')^{1/\alpha} : y'' \in Y_{2j+1}{}''\}$.
		❷ Cuckoo Hashing	Builds up Cuckoo hash table $\boldsymbol{\mathcal{C}}_{2j}$ and $\boldsymbol{\mathcal{C}}_{2j+1}$ with m rows. Then extracts these tables into one-dimensional vector ω_{2j} and ω_{2j+1}.
		❸ Windowing	For S_{Y_0}, selects the powers set φ and calculate the encrypted $[\omega_0{}^{\varphi_1}]_p$, $[\omega_0{}^{\varphi_2}]_p, \cdots, [\omega_0{}^{\varphi_{\lvert\varphi\rvert}}]_p$.
		❹ Differential Sharing	For the first two sets S_{Y_0}, S_{Y_1}, sends FHE ciphertexts $[\omega_0{}^{\varphi_1}]_p$, $[\omega_0{}^{\varphi_2}]_p, \cdots, [\omega_0{}^{\varphi_{\lvert\varphi\rvert}}]_p$ and $[\omega_1 - \omega_0]_p$ to *sender.* Else, sends $[\omega_{2j} - \omega_0]_p$ and $[\omega_{2j+1} - \omega_{2j}]_p$ to *sender*.
	❺ FHE Powers Generation		If j=0, generates all powers of ω_0 of the receiver according to Paterson-Stockmeyer algorithm.
	❻ Trident PoM		For each part i, generate polynomial set $H_{i,*}$ and detection set $Defect_{i,*}$ according to $[\omega_{2j} - \omega_0]_p$ and $[\omega_{2j+1} - \omega_{2j}]_p$. Batch $H_{i,*}$ into FHE plaintext polynomials, denoted as $P_i'[0], P_i'[1], \cdots, P_i'[n/\delta]$.
	❼ Trident Evaluation		For each P_i', evaluate $z_i \leftarrow \sum_{k=0}^{n/\delta} \omega_1{}^k \cdot P_i'[k]$ using Paterson-Stockmeyer algorithm. Send z and $Defect$ to *sender.*
		❽ FHE Decryption	Decrypts the ciphertexts and puts the intersection result into set S.

Fig. 4. Trident-PSI protocol. The protocol significantly reduces the communication by *Data Partitioning* step and *Differential Sharing* step, and FHE operations by *Trident Initialization*, *PoM* and *Evaluation*.

form such huge proportion. The 1st one is the explosion of the ciphertext. The more significant 2nd factor is the plenty of extra special operations introduced by FHE, which is not involved in the plaintext execution. As bottleneck on computation side, it is hence imperative to optimize FHE computation in PSI.

C. Acceleration Opportunities

According to the above analysis, FHE induces both significant communication and computation overhead in existing PSI protocols. Intuitively, using a specific FHE accelerator might be the solution. However, it only partially alleviates the computational bottleneck but does not solve it at the root. Firstly, although FHE induces significant computation overhead, FHE is only part of the protocol. APSI [10], MT-PSI [31], and Trident-PSI also include OPRF. Although the percentage of OPRF is only around 3% and 11% in each party, the actual time consumed is very large. For example, it consumes 3,315s (nearly 1 hour) for the receiver in APSI. In addition to OPRF, Trident-PSI also involves unique PoM operations. It is the computational bottleneck in Trident-PSI. OPRF and PoM operations are complex and involve irregular memory access patterns. If the hardware cannot support fast OPRF and PoM, PSI will still suffer from these bottlenecks. General-purpose computing platforms such as GPU are better suited for highly parallel and memory-simple floating-point operations, making them unsuitable for accelerating OPRF and PoM. Existing FHE accelerators [3], [18], [19], [27], [28], [33] only target FHE, so OPRF and PoM can only resort to CPU. Therefore, these accelerators are not suitable for PSI regarding both computation performance and efficiency, let alone the communication overhead imposed by FHE. Besides, existing FHE accelerators aim at general-purpose algorithm acceleration like CKKS, BFV, and TFHE. However, PSI imposes some restrictions on the choice of FHE schemes. APSI [10], ORI-PSI [6], and MT-PSI [31] all resort to BFV because they comply with the exactly precise decryption of the plaintext, while CKKS adheres to approximate decryption with acceptable precision loss, which makes it inefficient to implement PSI that requires a very precise intersection result, and TFHE is designed for bitwise logical operations, which makes it inefficient to implement arithmetic operations. Besides, PSI only involves leveled ciphertext multiplication, so bootstrapping is unnecessary as well.

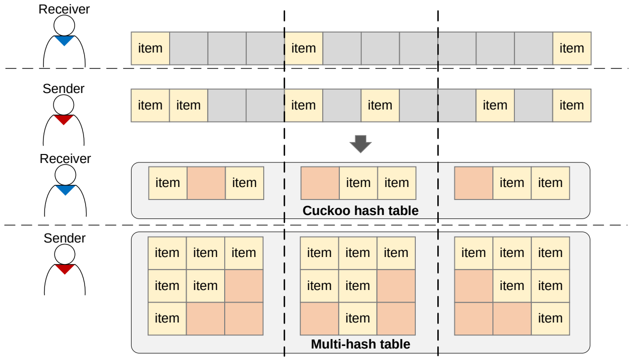
The oracle acceleration solution should consider two perspectives: (1) from the protocol level, re-architect the communication intensive procedures like FHE evaluation and windowing (Figure 2). We propose *Trident PoM*, *Differential Sharing*, etc. to optimize overhead in these part; (2) from the hardware level, design specific PSI accelerator for the optimized protocol to speed up its inner computation intensive procedures; we propose the Trident accelerator tailored for PSI, equipped with both customized PoM-FHE cores and OPRF cores. In total, it reduces the communication from 614 GB to 17 GB, and the computation from 159,745s to only 735s.

III. Trident-PSI Protocol

A. General Concept

The proposed Trident-PSI protocol, as shown in Figure 4, consists of two indispensable and associated phases - *online* and *offline*. In the *offline* phase, the data processed by the *receiver* and the *sender* is independent of the other party. Therefore, operations in *offline* phase only need to be executed once. The data obtained by both parties in *offline* phase can be used in current and future PSI *online* phases without modification.

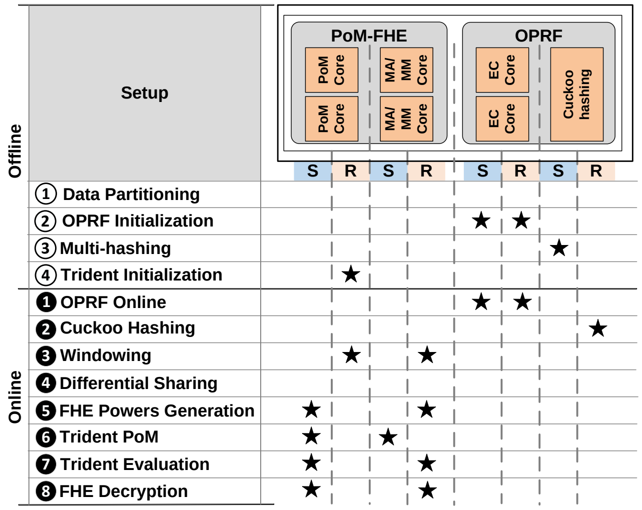
Trident proposes several new components compared to the APSI framework, including Data Partitioning (step ①), Trident



① Data Partition

Precondition: *Sender* and *Receiver* finish the Setup step.
Postcondition: In the following offline steps, if not specified, every set will be operated in each of the following offline steps.
Procedure:
(1) *Sender* and *Receiver* jointly agree on Γ pairs of integers. They are the lower bound and upper bound of Γ parts. Denotes i th part as S_i, with lower bound of L_i and upper bound of R_i. $R_i - L_i$ are the same with each other. Since data in *X* and *Y* are randomly uniform, these lower bound and upper bound information will not reveal any data information of either *Sender* or *Receiver*.
(2) *Sender* and *Receiver* partition *X* and *Y* respectly into Γ parts according to lower bound and upper bound information.

Fig. 5. Data Partitioning. We divide the data asset of both party into subsets, each of which has similar volume.



		S	R	S	R	S	R	S	R
Offline	Setup								
	① Data Partitioning								
	② OPRF Initialization					★	★		
	③ Multi-hashing							★	
	④ Trident Initialization		★						
Online	❶ OPRF Online					★	★		
	❷ Cuckoo Hashing								★
	❸ Windowing		★		★				
	❹ Differential Sharing								
	❺ FHE Powers Generation	★			★				
	❻ Trident PoM	★		★					
	❼ Trident Evaluation	★			★				
	❽ FHE Decryption	★			★				

Fig. 6. The steps accelerated by each computational unit of the Trident-accelerator.

Initialization (step ④), Differential Sharing (step ❹), Trident PoM (step ❻) and Trident Evaluation (step ❼).

The Data Partitioning step in the *offline* phase is proposed to segment the data assets into subsets in both *sender* and *receiver* as the basis to support other subsequent steps.

OPRF is introduced in both phases to improve protocol security against malicious *receiver*, according to Section II. We extract the parts of OPRF that do not require participation from the other party as OPRF Initialization and process it offline.

We adopt Cuckoo hashing & Multi-hashing mechanisms based on the blake2b cryptographic hashing used in APSI [10]. Multi-hashing can be completed without the participation of the *receiver*, so it can be processed offline. Cuckoo hashing is processed after the OPRF online step, so it can only be processed online.

Built upon Data Partitioning, the Differential Sharing step in the *online* phase is proposed to enable the transfer of the difference between the cryptographic hash digest of the first subset and the cryptographic hash digest of the subsequent subset, which reduces the communication significantly from the *receiver* to the *sender*.

The Trident PoM step is proposed to modify the polynomial of the *sender*. By enforcing this step, we can transform the burdensome FHE operations of the ciphertext into light-weight plaintext NTTs, which greatly decreases the computation time on the *sender* party.

In Trident Initialization, the *sender* pre-processing the polynomial used in Trident Evaluation. The Trident Evaluation step proposes a novel intersection method that embeds the PSI results of two subsets into one ciphertext polynomial, which directly obtains 50% communication reduction from the *sender* to the *receiver*.

The Trident-accelerator provides acceleration for both *offline* and *online* phases of Trident-PSI. Figure 6 shows the role and function of each computational unit of the Trident-accelerator. Step ① ❹ involve simple computations with little computational overhead. Therefore, Trident-accelerator does not support the acceleration of them. The PoM Core is implemented for accelerating steps ④ and ❻ of the Trident protocol. It is also reused to accelerate FHE operations in conjunction with the MA/MM Core. The EC Core and Cuckoo hashing are implemented to accelerate steps ② ③ and ❶ ❷, respectively.

B. Procedures

Data Partitioning: In FHE, one ciphertext polynomial with certain degree is able to encrypt multiple plaintext data. It is beneficial to generally reduce the ciphertext size. [10] indicates that for large-scale PSI, the *sender* has to read all data assets and maintain a huge multi-hash table, which demands more DRAM than the system can provide. This leads to excessive paging and a significant reduction in computation speed. One feasible solution from [10] is to partition the *sender*'s data asset into multiple subsets and let *receiver* compute the set intersection with all the segments of the *sender* party. However, this approach may cause the protocol parameters to exceed the reasonable range and may increase the communication from the *sender* to the *receiver*.

The aim of Data Partitioning is to divide the data asset of both parties into subsets with each one having the same data range. Therefore, the intersection of the two parties is computed in multiple rounds. In each round, the *receiver*'s subset only computes PSI with the corresponding subset of the *sender*, instead of with all the subsets. Figure 5 shows an example of Data Partitioning. In this example, the receiver and the sender each divide their items into three subsets and perform the intersection using the corresponding subsets.

The partition step is shown in Figure 5. The *Sender* and the *Receiver* partition X and Y into Γ subsets respectly. These subsets have the same data ranges, ensuring that the data in X and Y are uniformly partitioned. As stated in Section II, the data are uniformly distributed. Hence, after Data Partitioning, the distribution of each subset is consistent with that of the original data asset. Neither party can learn any information about the data distribution from the lower and upper bounds of each subset. Following the step of Data Partitioning, the parameters

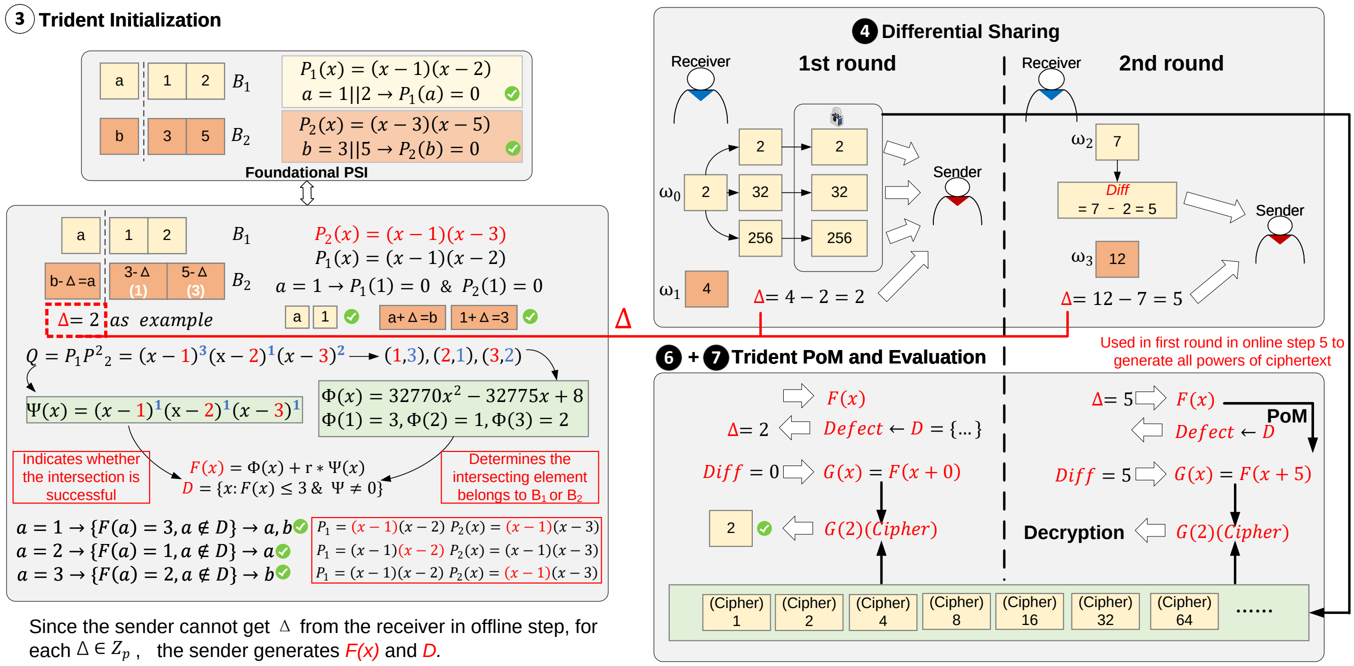

3 Trident Initialization
a
1
2
B_1
b
3
5
B_2
$P_1(x) = (x-1)(x-2)$
$a = 1||2 \rightarrow P_1(a) = 0$
$P_2(x) = (x-3)(x-5)$
$b = 3||5 \rightarrow P_2(b) = 0$
Foundational PSI
b- Δ =a
3- Δ (1)
5- Δ (3)
Δ= 2 as example
$P_2(x) = (x-1)(x-3)$
$P_1(x) = (x-1)(x-2)$
$a = 1 \rightarrow P_1(1) = 0$ & $P_2(1) = 0$
a+ Δ =b
1+ Δ =3
Δ
$Q = P_1P^2{}_2 = (x-1)^3(x-2)^1(x-3)^2 \rightarrow (1,3),(2,1),(3,2)$
$\Psi(x) = (x-1)^1(x-2)^1(x-3)^1$
$\Phi(x) = 32770x^2 - 32775x + 8$
$\Phi(1) = 3, \Phi(2) = 1, \Phi(3) = 2$
Indicates whether the intersection is successful
$F(x) = \Phi(x) + r * \Psi(x)$
$D = \{x: F(x) \leq 3 \;\&\; \Psi \neq 0\}$
Determines the intersecting element belongs to B_1 or B_2
$a = 1 \rightarrow \{F(a) = 3, a \notin D\} \rightarrow a, b$
$a = 2 \rightarrow \{F(a) = 1, a \notin D\} \rightarrow a$
$a = 3 \rightarrow \{F(a) = 2, a \notin D\} \rightarrow b$
$P_1 = (x-1)(x-2)$ $P_2(x) = (x-1)(x-3)$
$P_1 = (x-1)(x-2)$ $P_2(x) = (x-1)(x-3)$
$P_1 = (x-1)(x-2)$ $P_2(x) = (x-1)(x-3)$
Since the sender cannot get Δ from the receiver in offline step, for each $\Delta \in Z_p$, the sender generates F(x) and D.
4 Differential Sharing
Receiver
1st round
ω_0
2
32
256
ω_1
4
Sender
Δ= 4 − 2 = 2
Receiver
2nd round
ω_2
7
Diff = 7 - 2 = 5
ω_3
12
Sender
Δ= 12 − 7 = 5
Used in first round in online step 5 to generate all powers of ciphertext
6 + 7 Trident PoM and Evaluation
$F(x)$
Δ= 2
$Defect \leftarrow D = \{...\}$
$Diff = 0$
$G(x) = F(x+0)$
2
$G(2)(Cipher)$
Δ= 5
$F(x)$
PoM
$Defect \leftarrow D$
$Diff = 5$
$G(x) = F(x+5)$
Decryption
$G(2)(Cipher)$
(Cipher) 1
(Cipher) 2
(Cipher) 4
(Cipher) 8
(Cipher) 16
(Cipher) 32
(Cipher) 64
......

Fig. 7. The details of key steps - Trident Initialization, PoM, Evaluation and Differential Sharing.

for intersecting the subsets within each group will be identical and within reasonable range. Additionally, Data Partitioning serves as a fundamental step for subsequent optimization steps.

OPRF Initialization & OPRF Online: In each round during the OPRF Initialization and OPRF Online, the *sender* and *receiver* obtain S_X and S_Y, respectively, as described in Section II. The *sender* can calculate S_X independently, so the computation of S_X can be performed in OPRF Initialization. On the other hand, the *receiver* generates a secret key α and precomputes $Y' = \{H(y)^\alpha : y \in Y\}$ offline, which is sent to the *sender* during the online phase. Upon receiving Y', the *sender* computes $Y'' = \{(y')^\beta : y' \in Y'\}$ using secret key β and sends it back to the *receiver*. Finally, the *receiver* computes $S_Y = \{(y'')^{1/\alpha} : y'' \in Y''\} = \{H(y)^\beta : y \in Y\}$. The *receiver* is capable of computing Y' independently, therefore the computation of Y' can be performed in OPRF Initialization, while the remaining computation can be performed in OPRF Online. S_X and S_Y are used as inputs to perform the final PSI. To enhance the efficiency of OPRF Initialization and OPRF Online, we have designed an optimized $H(.)$ function on an elliptic curve over a hardware-friendly binary finite field.

Trident Initialization and Evaluation: In Section II, We specify that the transmission of homomorphic ciphertext from *sender* to *receiver* forms a significant proportion of the total communication, emphasizing the need to optimize this component. In order to prevent polynomial evaluations from reaching the noise overflow threshold, the *sender* splits each multi-hash table into δ parts. Finally, the *sender* transmits δ homomorphic ciphertexts to the *receiver*, resulting in a total of $\Gamma * \delta$ homomorphic ciphertexts being transmitted.

We propose Trident Initialization (offline step ③) and Trident Evaluation (online step ❼). In general, following the Data Partitioning step, it achieves two rounds of PSI result through one round of PSI only. The design concept of the Trident process is similar to keyword-PIR in APSI [10]. It queries the result of two rounds of PSI by evaluating an interpolated polynomial F. Figure 7 shows a naive example of Trident process. For ease of understanding, all Cuckoo hash tables and Multi-hash tables have only one row, and $\delta = 1$. The core of Trident Initialization and Evaluation is to construct new polynomials F and evaluate them. According to the construction method shown in Figure 7, we construct new polynomial F containing intersection information of subsets of two rounds. Informally, the value of $F(a)$ is 3, 1 or 2, which implies that intersection set contains both a and b, only a or only b respectively. Using this property, Trident computes two rounds of PSI in one time. Compared to the polynomial constructed in the foundational formulation, the degree of the new polynomial is doubled, without causing an overflow of the noise level.

Trident PoM (Polynomial Modification): Starting from the second round of PSI, Trident PoM (online step ❻) enables the *sender* to adjust F according to $Diff$ by applying Differential Sharing procedure and use ω_0 for polynomial evaluation. In practice, the Trident PoM procedure can be performed using NTT of length less than 256, so the Trident accelerator needs to effectively accelerate NTT with small length. As shown in Figure 7, the *sender* needs to evaluate $F(x+5)$ in the second round. The *sender* computes $G(x) = F(x+5)$, which can be achieved by applying NTT, and uses powers of the *receiver*'s ω_0 to evaluate $G(x)$ to get ciphertext result of $F(x+5)$. Since generating powers of ω_2 is unnecessary, the *sender* can skip step ❺ and avoids the complex FHE ciphertext multiplication operations. The modification of G for each row of each part can be computed by NTT with a time complexity of $O((n/\delta)log(n/\delta))$ and a space complexity of $O((n/\delta))$.

Windowing and FHE Powers Generation: We employ the Paterson Stockmeyer algorithm to construct the power set φ such that the *sender* can evaluate all powers of the *receiver*'s ω_0 with multiplication depth of at most 3-4 without bootstrapping, following the approach in [10].

TABLE III
INPUT AND OUTPUT OF *receiver* AND TWO FUNCTION. OUTPUT1 IS THE OUTPUT OF OPRF ONLINE PROCEDURE OF THE FIRST ROUND. OUTPUT2 AND OUTPUT3 ARE OUTPUTS OF THE DIFFERENTIAL SHARING PROCEDURE OF THE FIRST ROUND. OUTPUT4 IS THE OUTPUT OF THE OPRF ONLINE PROCEDURE OF THE SUBSEQUENT ROUND. OUTPUT5 IS THE OUTPUT OF THE DIFFERENTIAL SHARING PROCEDURE OF THE SUBSEQUENT ROUND. THE SYMBOLS USED ARE CONSISTENT WITH THOSE USED IN FIGURE 4.

Character	**Input**	$View_1$	$View_2$	$View_3$	$View_4$	$View_5$
receiver	X, Y, FHE params, OPRF params	$S_{y''_0}$, $S_{y''_1}$	$[\omega_0{}^{\varphi}]_p$	$[\omega_1-\omega_0]_p$	$S_{y''_{2*I}}$, $S_{y''_{2*I+1}}$	$[\omega_{2*I}-\omega_0]_p$, $[\omega_{2*I+1}-\omega_{2*I}]_p$
SIM_A	X, Y, FHE params, OPRF params	random values of size $\lvert S_{y''_0}\rvert+\lvert S_{y''_1}\rvert$	$[\omega_0{}^{\varphi}]_p$	$[\omega_1-\omega_0]_p$ from $I(\beta)$	random values of size $\lvert S_{y''_{2*I}}\rvert+\lvert S_{y''_{2*I+1}}\rvert$	$[\omega_{2*I}-\omega_0]_p$, $[\omega_{2*I+1}-\omega_{2*I}]_p$ from $I(\beta)$
SIM_B	X, FHE params, OPRF params	random values of size $\lvert S_{y''_0}\rvert+\lvert S_{y''_1}\rvert$	FHE encryptions of $\lvert\varphi\rvert$ random vectors on Z_p of size $\lvert\omega_0\rvert$	$[\omega_1-\omega_0]_p$ from $I(\beta)$	random values of size $\lvert S_{y''_{2*I}}\rvert+\lvert S_{y''_{2*I+1}}\rvert$	$[\omega_{2*I}-\omega_0]_p$, $[\omega_{2*I+1}-\omega_{2*I}]_p$ from $I(\beta)$

Differential Sharing: Based on Data Partitioning, we enroll Differential Sharing (formally described in the online step ❹ in Figure 4) to further reduce the communication overhead. Figure 7 shows an example of the Differential Sharing step in the same scenario as Trident Initialization, PoM and Evaluation. In general, the *receiver* only transmit ciphertext 2, 32, 256 and $\Delta = 2$ to the *sender* in the first round. In the subsequent rounds, the *receiver* computes the differential vector(denoted as $Diff$) and Δ, then sends them. We adopt cryptographic hashing as random function, so $Diff$ and Δ follow a uniform distribution over Z_p. Since S_{Y_*} follows uniform distribution, the *sender* is not able to get statistical information by analyzing $Diff$ and Δ. Besides, both $Diff$ and Δ are values that have been hashed by blake2b. $Diff$ and Δ cannot obtain their preimages by reversing blake2b. Therefore, the *sender* cannot learn useful information by analyzing $Diff$ and Δ.

C. Security Proof

Security model: Trident-PSI is secure against malicious *receiver* and semi-honest *sender*. This security setting is widely adopted by state-of-the-art PSI protocols such as [20] and [10].

Proof: We use the same proof idea in [6] to prove that Trident is secure against a malicious *receiver*. We use a general proof idea according to [22] to prove that Trident is secure against a semi-honest *sender*.

We prove that Trident is secure against malicious *receiver* as the first step. Since the malicious *receiver* may deviate from protocol to get advantage, the *receiver* can send arbitrary data to the *sender*. There is no difference between the behavior of the malicious *receiver* in Trident and in APSI because both protocol use OPRF as first online step. Therefore, we define the same ideal PSI functionality (see [6], Figure 3.) and adopt the same approach of proving Theorem 1 in [6].

We prove that Trident achieves simulation-based security against semi-honest *sender* as the second step. The proof idea is to construct a simulator to simulates the view of *sender*. Compared to protocols such as [20] and [10], the *sender*'s view in Trident includes additional access to $Diff$ and Δ. The sharing of $Diff$ and Δ reduces communication and computation overhead, but it also provides auxiliary information of $\{H(y) : y \in Y_*\}$ to the *sender*. However, according to Section III-B, $Diff$ and Δ are the differences of the values after the data is cryptographically hashed, so the *sender* cannot learn useful information from them.

For ease of writing, we define $[\omega_0{}^{\varphi}]_p = \{[\omega_0{}^{\varphi_1}]_p, [\omega_0{}^{\varphi_2}]_p, \ldots, [\omega_0{}^{\varphi_{|\varphi|}}]_p\}$. By $S_{y''_{2*I}}$, $S_{y''_{2*I+1}}$, $[\omega_{2*I}-\omega_0]_p$ and $[\omega_{2*I+1}-\omega_{2*I}]_p$, we mean $S_{y''_{2*i}}$, $S_{y''_{2*i+1}}$, $[\omega_{2*i}-\omega_0]_p$, $[\omega_{2*i+1}-\omega_{2*i}]_p$ respectively for each $i \in [2, \lceil\Gamma/2\rceil]$ in order.

To model the sharing of $Diff$, Δ in simulation-based security proof, we provide an interface I, which can query the ideal functionality for $Diff$ and Δ. The definition of I is as follows:

$I(\beta)$: Input β offered by simulator, compute $S_{Y_i} = \{H(y)^{\beta} : y \in Y_i\}$ for $i \in \Gamma$, and then compute ω_0, ω_1,..., ω_Γ. Return $[\omega_{2*I}-\omega_0]_p$ as $Diff$ and $[\omega_1-\omega_0]_p$, $[\omega_{2*I+1}-\omega_{2*I}]_p$ as Δ.

Since $Diff$ and Δ are public, both the simulator and the adversary can access the interface I. We define two simulators: SIM_A, SIM_B. Table III shows the input and view of two simulators. (1) Although the adversary can learn from interface I, by the assumption that One-More-Gap-Diffie-Hellman [23] is a hard problem, replacing $S_{y''_*}$ with uniformly random elements is computationally indistinguishable even though $\{H(y) : y \in Y_*\}$ is given. Therefore given auxiliary information of $\{H(y) : y \in Y_*\}$, $View_1$, $View_4$ given by SIM_A and *receiver* are still computationally indistinguishable with each other. (2) By the semantic security of FHE, the output of SIM_A and SIM_B are computationally indistinguishable with each other. According to (1) and (2), the view given by the *receiver* and SIM_B are computationally indistinguishable with each other. SIM_B is the desired simulator.

Therefore, Trident is secure against malicious *receiver* and semi-honest *sender*.

D. Discussion

Incremental Querying. The sharing of Diff and Δ can be conceptualized as a form of incremental querying. After OPRF, the *receiver* encrypts and transmits Cuckoo hash table of the first subset. For the remaining subsets, the *receiver* only needs to guide the *sender* on incrementally modifying this ciphertext to complete PSI, without re-encrypting and re-transmitting it.

Although this incremental query is protected by cryptographic hashing and it is secure according to Section III-C, it does indeed provide auxiliary information about data asset of the *receiver* to the *sender*. Its benefits are also evident, as it significantly reduces communication overhead. Differential Sharing (online step ❹) significantly reduces the amount of data transmitted from the *receiver* to the *sender*, from $\lceil\Gamma/2\rceil * (\Sigma * |\varphi| + n * 32bit)$ to $\Sigma * |\varphi| + (\Gamma - 1) * n * 32bit$ where Σ is the data size of the ciphertext after FHE and n is the number of plaintext data in a ciphertext polynomial. Besides, Differential Sharing replaces ciphertext-ciphertext multiplication with modification to polynomial set H. Trident Initialization (offline step ③) and Evaluation (online step ❼)

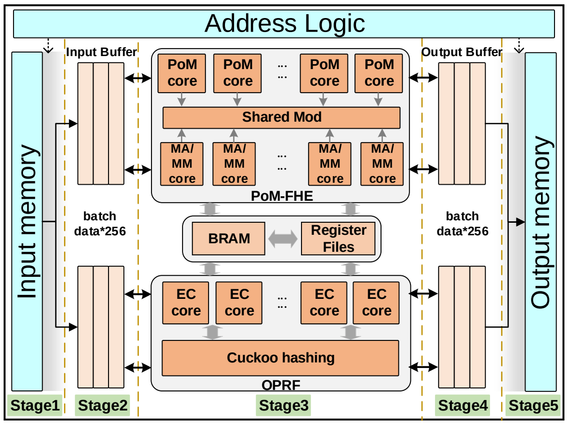


Fig. 8. The overall micro-architecture of the Trident-accelerator.

achieve two round of PSI result through one round of PSI, thus reducing the communication overhead from the *sender* to the *receiver* to half of the original.

Optimization on computation: Trident PoM (online step ❻) avoids complex FHE ciphertext multiplication according to Section III-B. It involves NTT of length less than 256. Therefore, the architecture of the hardware accelerator needs to effectively accelerate small length NTT.

IV. Trident Accelerator

A. *Overall Accelerator Architecture*

The overall architecture of the accelerator is shown in Figure 8. It primarily includes the computational cores (i.e., the elliptic curve (EC) cores and hash cores for OPRF, PoM cores, MA/MM cores and shared mod (SM) cores for FHE), the on-chip memory system (i.e., the input/output memory and intermediate data buffers), and the address access control logic. In the accelerator, the input memory is used to store the data from the off-chip memory system. To improve the data-level parallelism, each core (i.e., PoM core, MA/MM core and EC core) has its own buffer to store the polynomial coefficients from the input memory. In each cycle, 256 operands are loaded from the input memory to the core and written to the output buffer after computation. We use multiple BRAMs and register files as scratchpad to cache the intermediate results and support the parallel read/write of the required data for each computational core in our FPGA-based evaluations. As shown in Figure 8, the accelerator works in a 5-stage pipeline to maximize the parallelism:

❶ **Input.** The accelerator fetches the data from the off-chip memory system to the on-chip input memory. ❷ **Preprocess.** In this stage, the data are classified and transferred to the corresponding input buffers. ❸ **Execution.** This is the main stage of computing PSI. The OPRF cores and FHE cores work in collaboration, following the steps of the Trident protocol. ❹ **Result collection.** After the computation, each core writes the results to its corresponding output buffer. ❺ **Output.** In this stage, the computation results are wrapped up and are written back to the off-chip memory system.

In the design, the host directly transfers the data to the memory on the accelerator side via XDMA and PCIe Gen 4 interface. During the computation, the accelerator computes with the segment-wise data and hides the transfer delay by overlapping the transfer and the computation, since the input data of PSI protocol is well segmented. As the bandwidth of PCIe interface is 16 GB/s, the overlapping mechanism also ensures that the communication between the built-in memory of the FPGA platform and the host will not be a bottleneck.

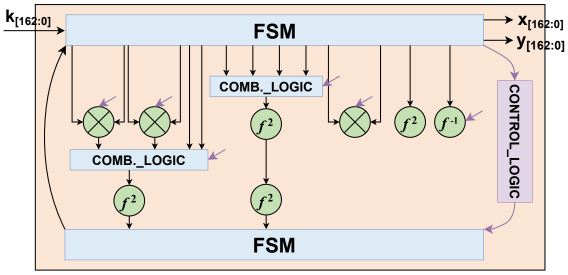


Fig. 9. EC core micro-architecture in the Trident-accelerator. The operators marked in green are the operators on binary finite fields.

B. *OPRF*

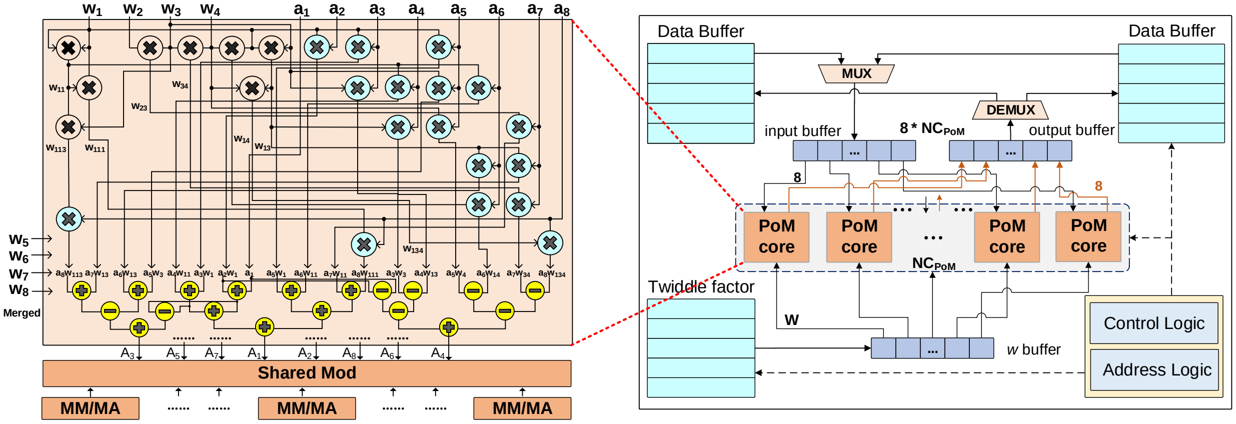
The basic operation of OPRF is the scalar multiplication of data points on elliptic curves (ECMult hereafter), and we accelerate the OPRF by instantiating the EC cores in the accelerator. Because of the independence of input data, multiple EC cores can process the elliptic-curve encryption in parallel. Cuckoo hashing is needed online, so we implement it conjunction with the EC cores, while we do not focus the offline Multi-hashing because the computation requires only once and can be completed with little overhead in the CPU.

EC Core: In order to avoid the division operation brought by affine coordinates [5], EC core uses projected coordinates to implement ECMult. The LD (Lopez&Dahab) coordinate [21] is selected, and the Montgomery Algorithm [16] is used to implement ECMult. The advantage is that the unique LD projection coordinate formula [25] for point doubling and point addition can be used. As shown in Figure 9, the input of each EC core is a 163-bits privacy key k, and the output is a single coordinate (x,y). In order to balance the high-frequency work and resource consumption of the system, we use the finite state machine (FSM in Figure 9) to schedule the delicate binary finite field operators: binary finite field multiplication (BMult hereafter), binary finite field square (BSqr hereafter), and binary finite field modular inverse (BModInv hereafter). We deploy three BMult units, four Bsqr units, and a high-speed BModInv unit in the EC core to balance the speed and resources.

Hashing: Trident-accelerator adopts blake2b cryptographic hashing as the hash function of Cuckoo hash, which is a fast and well-tested cryptographic hash function that mainly utilizes data rotation and multiplication operations. According to this feature, we implement full-pipelined blake2b operators by instantiating shift registers and DSPs on the FPGA.

C. *PoM-FHE*

PoM cores: PoM core is implemented for accelerating the step ④ and ❻ of Trident protocol. In the process, the *sender* party uses the NTT algorithm [11] to modify the intersection polynomial concurrently when it receives the next round of intersection messages from the *receiver* party. Similarly, the PoM core is also used to accelerate FHE process of Trident, which involves NTT as well. Since NTT is time-consuming and frequently used operation in PSI, the design of PoM cores significantly impact the performance of the accelerator.



(a) PoM Core (b) Access Pattern

Fig. 10. PoM-FHE micro-architecture in the Trident accelerator. The PoM cores support a parallelism of 256 to match the PoM operation exactly, with fewer iterations and a fully pipelined design to efficiently support the PoM operation in Trident-PSI.

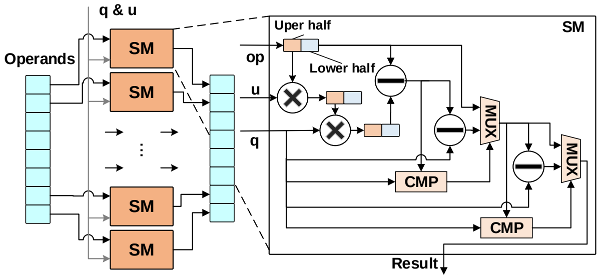


Fig. 11. Shared Mod. It hard codes the Barrett Reduction algorithm.

We improve the throughput of PoM core by merging multiple iterations of NTT into a single iteration. Using this method, one iteration can complete the computation process of multiple iterations that can improve the acceleration efficiency. We set the merging parameter to 3 in our design, which transforms the conventional 3-phase NTT into the 1-phase. As shown in Figure 10 (a), the PoM core takes 8 polynomial coefficients and the associate twiddle factors as input. We can see that only one modulo operation is required due to the single phase after merging. In addition, because of the exponential nature of the twiddle factors, we can take the newly generated twiddle factors as input (i.e., $w5 \sim w8$ in the figure), which can be pre-computed. Therefore, the PoM core costs fewer multipliers compared to the general handling process; only the blue ones in the Figure 10 (a) are involved. Fewer hardware resources are required to enable more PoM cores in the accelerator. We deploy 32 PoM cores to support the parallelism of 256, which fully matches the PoM operation of Trident-PSI.

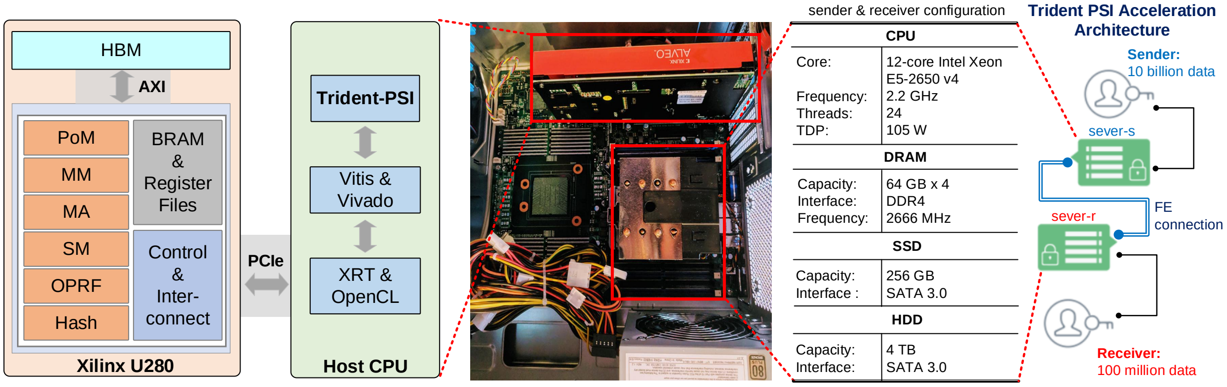
In Figure 10 (b), to support pipelined execution of PoM cores, we allocate BRAM blocks as the input buffer for each PoM core and use ping-pong operation to overlap the data read/write delay between NTT iterations. PoM cores read operands and twiddle factors from BRAMs, and write the results to the corresponding BRAMs under the control of address logic.

FHE cores (MA/MM): Since the PSI protocol requires a shallow multiplication depth and a small FHE parameter setting, the accelerator does not need to support the ciphertext refresh operation, i.e., bootstrapping. We only need three basic operators, i.e., ModMult (MM), ModAdd (MA) and NTT which can be accelerated by PoM cores mentioned above. The combination of these three operators supports all the required FHE operations in PSI, i.e. Homomorphic Multiplication (HMult), Homomorphic Addition (HAdd). HMult includes relinearization (or keyswitching) and modulus switching. In Table 3, We use HMult and HAdd for the comparison of computation performance. Notably, because we employ the RNS-based FHE scheme, and each RNS component modulus is 32 bits, the maximum bit width supported by all computing cores is a uniform 32 bits. For two polynomials c_0 and c_1, ModAdd returns a polynomial $c = c_0 + c_1$, where the operator "+" represents the element-wise addition of the two polynomials. Since FHE is performed on the "polynomial ring", the addition result equals to the modular addition, i.e., $c_0 + c_1 = (c_{0,i} + c_{1,i})\ mod\ q$, where q is the modulus. The modular multiplication (ModMult for clarity) operation of the polynomial is similar to ModAdd, which formally described as $c_0 \times c_1 = (c_{0,i} \times c_{1,i})\ mod\ q$. ModMult is more complex than ModAdd. Usually, modulo operation requires the division to get the quotient which is not FPGA friendly, so selecting a proper method for the modulo operation is crucial for accelerator design. Since the modulo operation is used not only in ModMult, we set up a dedicated modulo unit i.e., Shared Mod (SM) that can be time multiplexed by different computational cores to efficiently use hardware resources.

Shared Mod (SM): The modulo operation is the most complex mathematical operation in PSI, and the hardware overhead is relatively high, so we instantiate a shared modulo unit to reduce the consumption of FPGA resources. We employ the Barrett Reduction algorithm [14], which simplifies the modulo process by introducing the precomputed parameter u. As shown in Figure 11, the entire modulo process is accomplished using multipliers, adders and selectors.

D. Memory System

The on-FPGA memory system comprises register files and BRAMs. The accelerator also leverages the abundant off-FPGA bandwidth provided by HBM to accelerate the data movement. The HBM architecture involves two HBM2 stacks, and each stack has 16 channels. Each channel is 64 bits of data, and data bits can be transferred up to 1800 Mbps, which can provide a theoretical bandwidth of 460 GB/s. The initial data is loaded from the host DDR to the HBM of the U280 via the PCIe interface and sent to the register file or BRAM directly connected to the computation cores. The computation cores get the data from the scratchpad to complete the calculation



sender & receiver configuration

CPU	
Core:	12-core Intel Xeon E5-2650 v4
Frequency:	2.2 GHz
Threads:	24
TDP:	105 W
DRAM	
Capacity:	64 GB x 4
Interface:	DDR4
Frequency:	2666 MHz
SSD	
Capacity:	256 GB
Interface :	SATA 3.0
HDD	
Capacity:	4 TB
Interface:	SATA 3.0

Fig. 12. Trident acceleration architecture overview. We experiment the full system by instantiating the *sender* and *receiver* parties located in different places connected by 100M public network. The Trident accelerator is deployed on both parties performing the Trident-PSI protocol.

and write back. In Section V, we employ Xilinx Alveo U280 [32] to implement the Trident accelerator.

V. Evaluation

A. Experimental Setup

Platform. Trident is a practical PSI acceleration architecture. We implement both Trident-PSI in software and the Trident accelerator hardware, and we build a real-world experimental platform as shown in Figure 12. The accelerator is implemented using Xilinx U280 FPGA and deployed on both parties performing Trident-PSI. The host software interacts with the Xilinx Runtime (XRT) [2] environment and OpenCL framework to interact with U280 through PCIe. For ASIC implementation, we use a 7nm FinFET predictive process design kit [8], [9] and model the scratchpad as SRAMs by [29] to synthesize RTL implementation of Trident-accelerator ASIC.

PSI is adopted in a loan blacklist sharing scenario. The blacklist data includes identity information (such as ID numbers) and loan information (such as personal assets) of blacklisted users. The two parties involved in blacklist sharing encode identity information and loan information of each blacklisted user into a 64bit identifier. Each party holds 10 billion and 100 million identifiers, respectively, and aims to identify common users in their blacklists through loan blacklist sharing. We evaluate the end-to-end performance of the system over a 100M public network. The FHE algorithm we use is RNS-based BFV. The polynomial degree is 8192, plaintext modulus and chipertext modulus are 16 bits and 216 bits, and the parameter is 128-bits security. The parameter configuration of Trident-PSI is in consistent with the baseline for fairness.

Baseline. We compare Trident-PSI protocol with two baseline PSI protocols, namely ORI-PSI [6] and the state-of-the-art APSI [10] (see Table I). We evaluate performance of Trident-PSI and the two baselines on CPU (implemented with Microsoft SEAL Library [7]) and the state-of-the-art general-purpose FHE accelerator FAB [3]. FAB is the latest FPGA-based pure FHE accelerator, and we use single card version of FAB to compare with Trident-accelerator. We use FAB to prove that pure FHE accelerators are not suitable for accelerating PSI protocols.

B. PSI Performance

Table IV shows the performance comparison of ORI-PSI, APSI, and Trident-PSI on all the basic operations of PSI protocols. On CPU, benefit from the PoM, Trident-PSI performs fewer HMult and HAdd than ORI-PSI and APSI. In addition, the Trident-PSI significantly reduces the overhead of encryption and decryption operations. Hardware accelerators can effectively improve the execution efficiency of PSI, but there is an apparent gap between FAB [3] and PSI-specific Trident-accelerator, especially on our Trident-PSI protocol. The reasons are twofold: 1) FAB can not support the key operations of PSI, i.e., OPRF and Hash. Although the performance on FHE-related operations of the Trident-accelerator is close to FAB, the Trident-accelerator outperforms FAB by over 5× on the entire PSI protocols because the OPRF and Hash have to rely on CPU in FAB. 2) FAB has inferior performance on PoM. In Trident-PSI, PoM is a crucial procedure, in which the length of polynomials is less than 256. Although the NTT unit in FAB can accelerate PoM, the highly parallel NTT unit specifically designed for FHE cannot be fully exploited and thus cannot accelerate PoM efficiently. The PoM unit in Trident-accelerator has fully matched parallelism and the merging feature, so it can augment acceleration efficiency while fully utilizing hardware resources, and outperforms FAB by about 11×. Therefore, PSI necessitates a specialized accelerator and a generic FHE accelerator cannot optimize the PSI effectively.

Table V compares the communication volume of different PSI protocols. Trident-PSI optimizes the communication in two directions compared to the ORI-PSI and APSI, obtaining 21.6 × and 43.2 × the traffic decreasing, which benefits from protocol-level optimization, i.e., Differential Sharing and PoM.

C. Long-item PSI

Under the parameter setting of $\sigma = 128$, we compare the performance of APSI [10], MT-PSI [31], and Trident-PSI. In this long item situation, after OPRF, APSI slices each item in S_x and S_y into 4 shorter items and uses these shorter items for subsequent PSI steps. This results in a significant increase in communication overhead. MT-PSI uses the PoL technique to build links between each shorter item. The *receiver* only needs to encrypt and transmit the first shorter item to complete the PSI. MT-PSI also proposes a hybrid protocol that combines the PoL technique with the VBF technique. This protocol reduces offline preprocessing time at the cost of communication improvement. To compare the optimal online performance of each PSI protocol, we select MT-PSI with PoL only as our baseline. As shown in Table VI, MT-PSI achieves 4× decrease in communication overhead compared to APSI. The Trident protocol uses APSI's slicing method to handle the long item situation. Thanks to Differential Sharing and PoM, Trident

TABLE IV
PERFORMANCE BREAKDOWN OF PSI PROTOCOL ON CPU, FAB, AND TRIDENT-ACCELERATOR (FPGA). THE DATA OF THE TRIDENT-ACCELERATOR IS NORMALIZED TO CPU. SINCE FAB CAN NOT SUPPORT OPRF AND HASH, IT NEEDS RESORT TO THE CPU FOR THESE OPERATIONS.

Protocol / Stage	ORI-PSI [6]			APSI [10]			**Trident-PSI**		
	CPU	FAB [3]	Trident-accel.	CPU	FAB [3]	Trident-accel.	CPU	FAB [3]	Trident-accel.
OPRF & Hash (S,R)	7,192 s	7,192 s	**46.8 s (153×)**	6,950 s	6,950 s	**46.8 s (148×)**	7,153s	**7,153s**	**46.8 s (152×)**
PoM (S)	/	/	/	/	/	/	82,233 s	1,014 s	**403.9 s (203×)**
HMult & HAdd (S)	126,068 s	1,489 s	**1,433 s (87×)**	111,780 s	1,546 s	**1,483 s (75×)**	16,992 s	279.7 s	**279.7 s (60×)**
Encryption (R)	1,946 s	10.2 s	**8.8 s (221×)**	1,219 s	6.1 s	**5.3 s (230×)**	0.0198 s	0.0001 s	**0.00009 s (220×)**
Decryption (R)	24,536 s	112.2 s	**137.4 s (178×)**	24,127 s	112.2 s	**137.4 s (175×)**	370.8 s	3.5 s	**4.3 s (86×)**
Sum	159,744 s	8,766 s	**1,627 s (98×)**	144,077 s	8,819 s	**1,673 s (86×)**	106,750 s	8,451 s	**735 s (145×)**

TABLE V
THE COMMUNICATION PERFORMANCE OF PSI PROTOCOLS.

Protocol / Stage	ORI-PSI [6]	APSI [10]	**Trident-PSI (ours)**
R - S	102.9 GB	68.3 GB	4.75 GB **(21.6×)**
S - R	510.6 GB	510.1 GB	11.81 GB **(43.2×)**

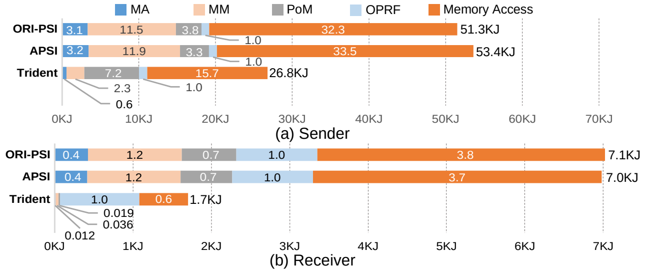
TABLE VI
COMMUNICATION & COMPUTATION PERFORMANCE ON CPU & TRIDENT-ACCELERATOR (FPGA) OF LONG-ITEM PSI PROTOCOLS.

Protocol / Stage	APSI [10]	MT-PSI [31]	**Trident-PSI (ours)**
Comm.	2289.6 GB	578.4 GB	42.4 GB **(54.0×)**
Comp. (CPU)	555,458 s	555,458 s	405,541 s **(1.4×)**
Comp. (Trident-accel.)	6551.6 s	6551.6 s	2799.6 s **(2.3×)**

achieves 54× decrease in communication overhead compared to APSI, and a 14× decrease compared to MT-PSI.

D. Energy

Energy Consumption and Breakdown. Besides the performance, we evaluate the energy consumption of two PSI baselines and Trident. Using the EDA tool Vivado, we obtain hardware power consumption and combine it with "wall clock" computation time to calculate energy usage. As shown in Figure 13, Trident consumes less energy than ORI-PSI and APSI by significantly reducing FHE operations including HMult, HAdd, encryption, and decryption. The memory access takes up most of the share. Figure 13 shows the energy consumption of the Trident accelerator across the three PSI protocols, analyzing hardware units on the *sender* and *receiver* sides. On the *sender* side, the energy consumption for OPRF is the same across protocols, as they are computationally indistinguishable at this stage. For MA, MM and memory access, the energy consumption of Trident is significantly reduced compared to APSI and ORI-PSI due to the fact that the Trident Initialization, PoM and Evaluation of the protocol greatly reduces the complexity of the intersection polynomial computation at the *sender* side, which reduces the energy consumption of the computational unit and also reduces accesses, thus bringing the benefit of access to energy consumption. It is worth noting that Trident incurs approximately double the PoM energy overhead compared to the other two protocols, as Trident PoM uses PoM core to modify the polynomial, leading to a higher energy cost. On the *receiver* side, except for the same OPRF computation, Trident obtains a significant energy advantage on all other hardware units, especially on the computation related units, i.e. MA, MM and PoM, with over 30× energy reduction due to Differential Sharing and Trident PoM. The significant power advantage validates the excellent performance of Trident over other PSI protocols in hardware-software co-design.



(a) Sender

(b) Receiver

Fig. 13. Energy consumption and breakdown.

TABLE VII
EFFICIENCY EFFICIENCY. WE USE ENERGY DELAY PRODUCT (EDP) AS THE METRIC (ENERGY × TIME). LOWER IS BETTER.

Stage / System	Sender		Receiver	
	EDP	improvement	EDP	improvement
ORI-PSI [6]	1.77×10^{12}	1×	9.32×10^{10}	1×
APSI [10]	1.40×10^{12}	1.26×	8.63×10^{10}	1.08×
Trident	$\mathbf{1.89\times 10^{7}}$	**93,651×**	$\mathbf{4.8\times 10^{4}}$	**1,102,960×**

Energy Efficiency. In addition to the energy consumption, we also use Energy Delay Product (EDP hereafter) metric to evaluate the energy efficiency. Similar to the energy consumption, the EDP performance is also divided into two parts, i.e., the *receiver* and the *sender*. Table VII lists the comparison result of three PSI system, Trident's energy efficiency is far superior to the other PSI systems, which outperforms APSI by tens and hundreds of thousands on both sides, respectively.

E. Design Space Exploration

Data Asset Scaling. We evaluate Trident with four different data volumes that vary in size and ratio between the *sender*

TABLE VIII
DATA ASSET SCALING ANALYSIS.

Stage S / R	receiver	Comm.	sender
2^{27} / 2^{20}	28.7 s	200.2 MB	828.3 s
2^{30} / 2^{23}	228.4 s **(8.0×)**	1,065.4 MB **(5.3×)**	6,466 s **(7.8×)**
2^{34} / 2^{27}	3,651 s **(127.2×)**	16.6 GB **(84.9×)**	103,099 s **(124.5×)**
2^{36} / 2^{28}	6,467 s **(225.3×)**	48.7 GB **(249.1×)**	411,019 s **(496.2×)**

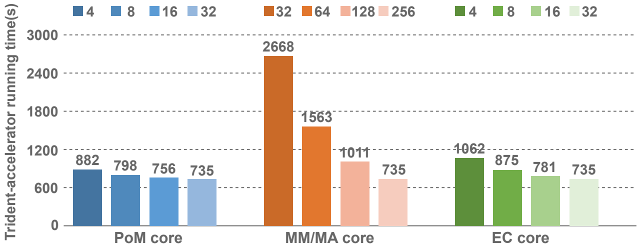


Fig. 14. Computation core scaling analysis. We analyze the performance impact by scaling three computing cores in Trident.

and the *receiver*. As shown in Table VIII, under the same data ratio, Trident's processing time grows approximately linearly with the data size. When the ratio of data between *receiver* and *sender* changes, i.e., the last item of Table VIII, Trident's computation time does not increase in proportion to the sum of data growth at both parties. It indicates that Trident's data partitioning strategy can well divide the intersection subset without severely affecting the system performance by inefficient partition. The result proves that Trident can stably support different PSI data scales.

Computation Core Scaling. We explore the performance scaling with three key computation cores, i.e., PoM cores, MM/MA cores and OPRF cores. As shown in Figure 14, intuitively, more computation cores bring a faster PSI performance. We combine the impact of different cores on the system performance and thus allocate 32 PoM cores, 256 MM/MA cores and 32 OPRF cores in a balanced manner to maximize the system performance and FPGA resource utilization.

F. Ablation Study

Trident is a co-designed PSI system. In this subsection, we carry out the ablation study in protocol and accelerator. We separately evaluate the optimization of the Trident compared to the current PSI protocols, i.e., ORI-PSI and APSI, in terms of communication and computation. We also analyze the performance improvement of two key optimization methods. In terms of the accelerator, we compare Trident with Trident + CPU to show hardware accelerator effect in whole system.

Data Partitioning, Differential Sharing and Trident PoM is a tightly coupled protocol steps, so we use DPDP to denote them. Similarly, we use TIE to denote the coupled Trident Initialization and Evaluation steps. Table IX(a) exhibits the impact of DPDP and TIE on PSI efficiency at the software level. We have 3 scenarios in total, i.e., No DPDP & TIE, DPDP only and DPDP + TIE. Compared to No DPDP & TIE, The DPDP achieves 7.3× and 1.1× speedup on the *receiver* and *sender* parties, respectively, and reduces the communication volume by 23.8× the original. DPDP + TIE further improves PSI efficiency based on DPDP. The two-step optimization significantly reduces the amount of communication and decreases the computational overhead to a certain extent.

TABLE IX
ABLATION STUDY. IN ASSOCIATED WITH TABLE V, DPDP + TIE REDUCES COMMUNICATION, PRIMARILY INTRODUCED BY FHE.

(a) Protocol

Stage Method	receiver	Comm.	sender
No DPDP & TIE	28,659 s	578.4 GB	115,418 s
DPDP only	3,941 s **(7.3×)**	24.3 GB **(23.8×)**	108,020 s **(1.1×)**
DPDP + TIE	3,651 s **(7.9×)**	16.6 GB **(34.8×)**	103,099 s **(1.1×)**

(b) Accelerator

Stage System	receiver	sender
No OPRF & PoM & FHE	3,651 s	103,099 s
FHE only	3,284 s **(1.1×)**	86,449 s **(1.2×)**
FHE + PoM	3,284 s **(1.1×)**	4,950 s **(20.8×)**
OPRF+PoM+FHE	28 s **(130.4×)**	707 s **(145.8×)**

TABLE X
FPGA RESOURCE UTILIZATION.

Resource Module	**LUT (k)**	**FF (k)**	**DSP**	**BRAM**
MA	69	35	0	334
PoM	233	110	2,944	928
MM	15	17	1,024	334
SM	78	60	1,792	0
OPRF	617	171	704	192

The hardware accelerator targets the high computation overhead, which cannot be solved by protocol optimization. As shown in Table IX(b), Trident significantly reduces the computational overhead of the system. Compared to Trident running on the CPU, the FHE only achieves the speedup ratios of 1.1× and 1.2× on the *receiver* and *sender* parties, and the FHE + PoM obtains 20.8× performance improvement in the *sender* party, which prove the importance of accelerating PoM. By accelerating all the Trident procedures, i.e., OPRF, PoM, and FHE, we achieve the speedup of 130.4× and 145.8×, which further demonstrates that pure FHE-specific accelerators cannot perform well facing PSI acceleration.

G. Accelerator Resource Utilization

The U280 is equipped with the Xilinx xcvu37p device. Table X exhibits the resource utilization. The SM cores and PoM cores obviously consume more DSP resources because there are many multiplication operations in PoM and modular arithmetic. The OPRF module uses more LUTs and fewer DSP resources due to the fact that the module does not contain multiplication operations. In addition, the MA cores consume the least resources as a result of its simple circuit logic. The design of the time-share modular unit SM enables the accelerator efficiently uses the limited FPGA resource to achieve a better performance.

H. ASIC Implementation

We implement our components in RTL and synthesized them in a commercial 7 nm process using state-of-the-art

TABLE XI
AREA AND POWER BREAKDOWN. THE ASIC IMPLEMENTATION IS RUNNING AT 1 GHZ AND UNDER THE 7NM TECHNOLOGY NODE.

Component	**Area [mm^2]**	**TDP [W]**
MA	0.025	0.092
MM	0.247	0.87
PoM	0.451	0.39
EC	0.208	0.196
Hash	0.036	0.063
NoC	0.5	0.66
Scratchpad SRAM (Cap. 8.6 MB / BW 3.9 TB/s)	3.7	0.32
HBM (2 × HBM2E) (Cap. 16 GB / BW 1 TB/s)	29.6	31.8
Sum	34.8	34.4

TABLE XII
PSI PERFORMANCE ON ASIC.

Platform	ORI-PSI	APSI	Trident-PSI
Trident-accel. (**FPGA**)	1,627 s	1,673 s	735 s
Trident-accel. (**ASIC**)	439 s	452 s	221 s

tools including the commercial SRAM compiler. Compared to FPGA implementation, we add an HBM memory to improve the bandwidth support, which can provide a total of 1 TB/s off-chip memory bandwidth. We use [17], [24] to modeled the HBM memory. The volume of the scratchpad is about 8 MB and can provide an on-chip bandwidth of 3.9 TB/s and we implement the NoC as a multiplexer network. The area costs and peak power of each hardware part are listed in Table XI. The HBM and SRAM account for the majority of the share. The Trident accelerator is sized 34.8 mm^2 and has the power consumption of 34.4 W. As shown in Table XII, benefit from the larger bandwidth of the HBM as well as the running frequency, the ASIC implementation further improves the acceleration of PSI by more than 3× compared to the FPGA-based Trident-accelerator.

VI. CONCLUSION

In this paper, we present a software/hardware co-design solution for PSI acceleration. By investigating the execution of PSI, we observe that the PSI performance suffers from the large communication volume and high computation intensity. Based on this observation, we propose a high-performance PSI acceleration system named "Trident." Trident includes a novel optimized PSI protocol called "Trident-PSI," which significantly reduces the communication overhead, and a Trident-specific hardware accelerator. We evaluate the Trident on the practical FPGA device and ASIC. The results demonstrate the effectiveness of the novel PSI acceleration architecture, i.e., Trident, which significantly improves both communication and computation performance. In Trident, FHE polynomial evaluation remains one of the computational overhead bottlenecks. This can be mitigated by optimizing relinearization insertion to reduce relinearization times. Currently, Trident-accelerator design has HMult fixed to insert relinearization immediately after ciphertext multiplication. This disables relinearization optimization in FHE polynomial evaluation. Therefore, adjusting the Trident-accelerator to be more suitable for relinearization insertion and designing specific relinearization insertion strategy is an improvement direction for the software-hardware co-design of Trident. Additionally, the Trident PoM step is used to replace the FHE Powers Generation procedure, thereby avoiding complex FHE ciphertext multiplications. However, PoM also introduces significant computational overhead due to its frequent use of NTT. Further acceleration of PoM from both algorithmic and hardware perspectives is also a future improvement direction for Trident. We hope this work can inspire new ideas for future PSI acceleration combining with the protocol optimization.

REFERENCES

[1] Datatrust. [Online]. Available: https://dp.alibaba.com/product/datatrust

[2] "Xilinxruntime rev:2021.1," https://www.xilinx.com/products/design-tools/vitis/xrt.html.

[3] 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.

[4] J. Ali. (2018) Validating leaked passwords with k-anonymity. [Online]. Available: https://blog.cloudflare.com/validating-leaked-passwords-with-k-anonymity/

[5] S. Atay, A. Koltuksuz, H. Hisil, and S. Eren, "Computational cost analysis of elliptic curve arithmetic," in *2006 International Conference on Hybrid Information Technology*, vol. 1. IEEE, 2006, pp. 578–582.

[6] 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.

[7] H. Chen, K. Laine, and R. Player, "Simple encrypted arithmetic library-seal v2. 1," in *International conference on financial cryptography and data security*. Springer, 2017, pp. 3–18.

[8] L. T. Clark, V. Vashishtha, D. M. Harris, S. Dietrich, and Z. Wang, "Design flows and collateral for the asap7 7nm finfet predictive process design kit," in *2017 IEEE international conference on microelectronic systems education (MSE)*. IEEE, 2017, pp. 1–4.

[9] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "Asap7: A 7-nm finfet predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.

[10] 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.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2022.

[12] D. Demmler, P. Rindal, M. Rosulek, and N. Trieu, "Pir-psi: scaling private contact discovery," *Cryptology ePrint Archive*, 2018.

[13] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold, "Keyword search and oblivious pseudorandom functions," in *Theory of Cryptography: Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005. Proceedings 2*. Springer, 2005, pp. 303–324.

[14] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to elliptic curve cryptography*. Springer Science & Business Media, 2006.

[15] C. Hazay and Y. Lindell, "Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries," in *Theory of Cryptography: Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008. Proceedings 5*. Springer, 2008, pp. 155–175.

[16] C. H. C. Huang, J. L. J. Lai, J. R. J. Ren, and Q. Z. Q. Zhang, "Scalable elliptic curve encryption processor for portable application," in *ASIC, 2003. Proceedings. 5th International Conference on*, vol. 2. IEEE, 2003, pp. 1312–1316.

[17] N. P. Jouppi, D. H. Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma *et al.*, "Ten lessons from three generations shaped google's tpuv4i: Industrial product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 1–14.

[18] J. Kim, G. Lee, S. Kim, G. Sohn, J. Kim, M. Rhu, and J. H. Ahn, "Ark: Fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse," *arXiv preprint arXiv:2205.00922*, 2022.

[19] S. Kim, J. Kim, M. J. Kim, W. Jung, J. Kim, M. Rhu, and J. H. Ahn, "Bts: An accelerator for bootstrappable fully homomorphic encryption," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 711–725.

[20] Á. Kiss, J. Liu, T. Schneider, N. Asokan, and B. Pinkas, "Private set intersection for unequal set sizes with mobile applications," *Cryptology ePrint Archive*, 2017.

[21] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of computation*, vol. 48, no. 177, pp. 203–209, 1987.

[22] Y. Lindell, "How to simulate it–a tutorial on the simulation proof technique," *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich*, pp. 277–346, 2017.

[23] C. Meadows, "A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party," in *1986 IEEE Symposium on Security and Privacy*. IEEE, 1986, pp. 134–134.

[24] M. O'Connor, N. Chatterjee, D. Lee, J. Wilson, A. Agrawal, S. W. Keckler, and W. J. Dally, "Fine-grained dram: Energy-efficient dram for extreme bandwidth systems," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017, pp. 41–54.

[25] S. Okada, N. Torii, K. Itoh, and M. Takenaka, "Implementation of elliptic curve cryptographic coprocessor over gf (2ˆ m) on an fpga," in *CHES*, 2000, pp. 25–40.

[26] J. Pullman, K. Thomas, and E. Bursztein, "Protect your accounts from data breaches with password checkup," 2019.

[27] M. S. Riazi, K. Laine, B. Pelton, and W. Dai, "Heax: An architecture for computing on encrypted data," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 1295–1309.

[28] 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 *ISCA*, 2022, pp. 173–187.

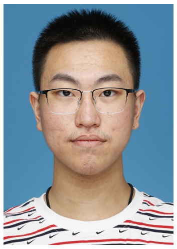
[29] A. Shafaei, Y. Wang, X. Lin, and M. Pedram, "Fincacti: Architectural analysis and modeling of caches with deeply-scaled finfet devices," in *2014 IEEE Computer Society Annual Symposium on VLSI*. IEEE, 2014, pp. 290–295.

[30] L. Shen, X. Chen, D. Wang, B. Fang, and Y. Dong, "Efficient and private set intersection of human genomes," in *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 2018, pp. 761–764.

[31] M. Wu and T. H. Yuen, "Efficient unbalanced private set intersection cardinality and user-friendly privacy-preserving contact tracing," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 283–300.

[32] Xilinx, "Product brief of smartssd," https://www.xilinx.com/products/boards-and-kits/alveo/u280.html.

[33] 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.



Jinkai Zhang is currently pursuing the Doctoral degree with the Institute of Computing Technology, Chinese Academy of Sciences, University of Chinese Academy of Sciences, Beijing, China. His research interests include fully homomorphic encryption application, fully homomorphic encryption acceleration, and fully homomorphic encryption compiler.

Yinghao Yang is currently pursuing the Doctoral degree with the Institute of Computing Technology, Chinese Academy of Sciences, University of Chinese Academy of Sciences, Beijing, China. His research interests include fully homomorphic encryption acceleration, FPGA accelerator design, and fully homomorphic processor design.

Zhe Zhou received the bachelor's degree from the School of Communications and Information Engineering, Nanjing University of Posts and Telecommunications, Nanjing, China, in 2021. He is currently working toward the PH.D. degree with the school of Cyber Science and Engineering, Southeast University, Nanjing. His main research interests include computer architecture and security and privacy preseving computing architecture.

Zhicheng Hu received the B.E. degree in microelectronics from University of Electronic Science and Technology of China, Chengdu and is pursuing the M.S. degree under the guidance of Prof Liang Chang from the Department of Internet of Things Engineering, School of Information and Communication Engineering. His current research interests include energy-efficient deep-neural-network architectures/accelerators and efficient deep learning based algorithm for hardware processing.

Xin Zhao received the B.E. degree in Electronic Engineering from China University of Mining and Technology, Xuzhou, China, in 2021. He is currently pursuing the M.S. degree under the guidance of Prof Liang Chang from the Department of Internet of things Engineering, School of information and Communication Engineering, University of Electronic Science and Technology of China. His research interest contains computing-in-memory architecture and super-resolution hardware architecture.

Liang Chang (M'19) received the Ph.D. and M.S. degrees from Beihang University in 2019 and 2014, respectively. He received a double B.S. degree from the Chengdu University of Information and Technology and the University of Electronic Science and Technology of China 2011. He was an engineer and senior engineer in China Glorun Technology (Beijing, China) and AMD China (Beijing, China) during 2012-2015. From 2022, he was a visiting scholar at HKUST. He has co-authored over 50 scientific papers, including IEEE International Solid-State Circuits Conference (2021, 2023, 2024), MICRO (2021), IEEE TCAS-I (2020-2024), IEEE TVLSI (2019, 2024), IEEE TC (2019), IEEE TCAD, etc. His research interests include computing in emerging nonvolatile memory, advanced memory-centric computer architecture, and AI processors for intelligent detection. He is the Regular Reviewer of the IEEE JSSC, IEEE TCAS-I/II, IEEE TBioCAS, IEEE TCAD, IEEE TC, IEEE TVLSI, etc.

Hang Lu received B.S. and M.S. degrees in Electronic Information Engineering from the Beijing University of Aeronautics and Astronautics, Beijing, China, in 2008 and 2011, respectively, and a Ph.D. degree in computer science from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, China, in 2015. He is currently an Associate Professor and a Master Tutor with the ICT, CAS, University of Chinese Academy of Sciences, Beijing, China. He is also a Research Scientist with Shanghai Innovation Center for Processor Technologies. His research interests include fully homomorphic encryption acceleration, FPGA accelerator design, and fully homomorphic processor design. Dr.Lu is a member of the Youth Innovation Promotion Association of CAS, and the New Best Star of ICT.

Xiaowei Li (Senior Member, IEEE) received the B.Eng. and M.Eng. degrees in computer science from the Hefei University of Technology in 1985 and 1988, respectively, and the Ph.D. degree in computer science from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS) in 1991. In 2000, he joined ICT as a Professor, where he is currently the Deputy Director of the State Key Laboratory of Computer Architecture. He has coauthored over 280 papers in journals and international conferences, alongside holding 60 patents and 30 software copyrights. His research interests include VLSI testing, design for testability, design verification, dependable computing, and wireless sensor networks. He services as an Associate Editor for the Journal of Computer Science and Technology, the Journal of Low Power Electronics, the Journal of Electronic Testing: Theory and Applications, and the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS.