Summation-based Private Segmented Membership Test from Threshold-Fully Homomorphic Encryption

Nirajan Koirala
University of Notre Dame
Notre Dame, Indiana, USA
nkoirala@nd.edu

Jonathan Takeshita
University of Notre Dame
Notre Dame, Indiana, USA
jtakeshi@nd.edu

Jeremy Stevens
University of Notre Dame
Notre Dame, Indiana, USA
jsteve22@nd.edu

Taeho Jung
University of Notre Dame
Notre Dame, Indiana, USA
tjung@nd.edu

ABSTRACT

In many real-world scenarios, there are cases where a client wishes to check if a data element they hold is included in a set segmented across a large number of data holders. To protect user privacy, the client's query and the data holders' sets should remain encrypted throughout the whole process. Prior work on Private Set Intersection (PSI), Multi-Party PSI (MPSI), Private Membership Test (PMT), and Oblivious RAM (ORAM) falls short in this scenario in many ways. They either require data holders to possess the sets in plaintext, incur prohibitively high latency for aggregating results from a large number of data holders, leak the information about the party holding the intersection element, or induce a high false positive.

This paper introduces the primitive of a Private Segmented Membership Test (PSMT). We give a basic construction of a protocol to solve PSMT using a threshold variant of approximate-arithmetic homomorphic encryption and show how to overcome existing challenges to construct a PSMT protocol without leaking information about the party holding the intersection element or false positives for a large number of data holders ensuring IND-CPA^D security. Our novel approach is superior to existing state-of-the-art approaches in scalability with regard to the number of supported data holders. This is enabled by a novel summation-based homomorphic membership check rather than a product-based one, as well as various novel ideas addressing technical challenges. Our PSMT protocol supports many more parties (up to 4096 in experiments) compared to prior related work that supports only around 100 parties efficiently. Our experimental evaluation shows that our method's aggregation of results from data holders can run in 92.5s for 1024 data holders and a set size of 2^{25}, and our method's overhead increases very slowly with the increasing number of senders. We also compare our PSMT protocol to other state-of-the-art PSI and MPSI protocols and discuss our improvements in usability with a better privacy model and a larger number of parties.

Proceedings on Privacy Enhancing Technologies YYYY(X), 1–17
© YYYY Copyright held by the owner/author(s).
https://doi.org/XXXXXXX.XXXXXXX

KEYWORDS

Multi-party private set intersection; Private membership test; Fully homomorphic encryption

1 INTRODUCTION

Privacy concerns often limit the collaboration of entities in the case where each entity has private data that must be shared for joint usage. In many cases, the private data is generated and stored in a distributed manner, and methods of sharing data privately in such scenarios open avenues for new applications. There are many real-world scenarios where this problem needs to be solved.

One example of this is the case where federal tax authorities want to learn whether any suspected tax evaders maintain accounts in both domestic and foreign banks that might be under scrutiny and, only if so, obtain their account records and details. The banks' locations in different jurisdictions prohibit the disclosure of account holders, and the tax authorities cannot openly divulge their list of suspects. Institutions under such scrutiny want to collaborate anonymously to avoid any bad publicity for being linked to a tax fraud operation. In many cases, these institutions are willing to collaborate with the tax authorities [61], and they themselves also wish to exercise rigorous scrutiny when extending loans to new and existing customers to mitigate potential risks. However, many financial privacy laws [85] prohibit banks from revealing customer data to third parties without consent. There are currently over 4,700 FDIC-insured banks in the United States. When a customer applies for a loan with one of them, collaboration and sharing of existing fraud lists can make the decision process significantly more trustworthy. In such cases, banks do not wish to share their private data on fradulent activities, and they do not even wish to disclose whether a queried customer is on their "fraud watchlist" due to various privacy and legal concerns. Such a secure membership query scenario could also include credit card companies, tax-collection agencies, and similar entities, necessitating the involvement of many parties in the decision-making process. A data-sharing protocol that allows queriers to learn only whether a queried entity exists in distributed fraud lists would help such institutions examine a person's credibility beyond the nation.

As another example, many government agencies (e.g., FBI, CIA) maintain sensitive lists of secret agents or watchlists distributed across its multiple divisions and branches. Data sharing for identity verification, background checks, security clearance, or watchlist

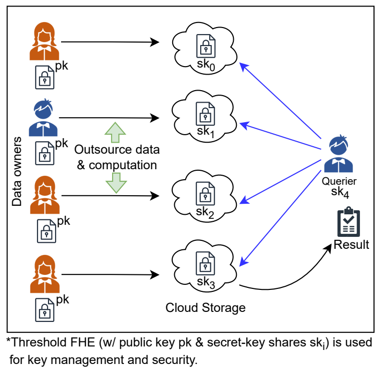

Data owners
pk
pk
pk
pk
Outsource data & computation
sk0
sk1
sk2
sk3
Cloud Storage
Querier sk4
Result
*Threshold FHE (w/ public key pk & secret-key shares ski) is used for key management and security.

Figure 1: Conceptual illustration of Private Segmented Membership Test (PSMT)

screening requires many government entities to work together. For example, a querier may want to verify an individual's presence in these entities' databases without inferring any membership details to verify the identity or the background. Doing so over distributed databases is nontrivial due to the challenges of sharing personally identifiable information (PII). It is desirable and even imperative that the records be stored in encrypted forms across a large number of distributed servers such that each record is under strict security/privacy protection (e.g., DHS Use Cases [86]). Similarly, the auto and medical insurance industries involve highly distributed records, and a protocol to query such databases can minimize risks.

In all the examples above, we face a problem where the number of entities involved in data-sharing applications can be substantial, e.g., thousands of entities when dealing with tax fraud [96]. Furthermore, datasets are updated frequently, and regulations force such datasets containing PII to be stored and operated with strong data security guarantees [37]. These examples all underscore the growing need for privacy-preserving set intersection protocols with a substantial number of dataset holders. There is a need to perform queries by testing the membership of a client's element within distributed datasets without leaking information about which set the intersection came from. We call this property *provenance privacy*, which refers to the confidentiality of from which party the intersection comes from. These datasets are updated frequently and held by multiple distinct parties, which should be strictly protected to ensure individual privacy. For this purpose, storing and using records in encrypted form can safeguard against data breach attacks or insider threats. This allows data holders (e.g., Amazon AWS or Microsoft Azure) to hold their customers' (data owners') data in encrypted form to comply with privacy regulations. These data holders store and operate on the encrypted data provided by their clients or data owners (e.g., financial institutions, hospitals, tax-collecting, and law-enforcement agencies) on their behalf. This scenario is different from those considered by privacy-preserving techniques such as PSI, MPSI, PMT or ORAM (detailed below and in Section 2) in multiple ways. We term this problem in such a scenario with multiple data holders *Private Segmented Membership Test* (PSMT). A conceptual illustration of PSMT is given in Figure 1.

Existing approaches, such as private set intersection (PSI), fall short in numerous ways for those scenarios. PSI enables two parties (referred to as *receiver* and *sender* hereafter by convention [43, 82, 93]) to compute set intersections without revealing any additional information. While existing PSI protocols can address the PSMT, they require dataset holders to access the elements of their sets in plaintext format [23] to enable various optimizations for polynomial interpolation, and they do not scale well with a high number of parties. Thus, for situations with encrypted databases with sensitive information such as medical, financial, or criminal records, PSI is not suitable. Generic multiparty-PSI (MPSI) protocols like [11, 18, 71, 84, 107, 109] are also less suitable for efficiently solving PSMT due to a high number of interactions (in OT-based), privacy concerns against the senders, high aggregation runtime, bandwidth or storage needs. Additionally, Private Membership Test (PMT) protocols [97], optimized for single elements, require unencrypted databases, may produce high false positives due to filters for representing the database [33], and are not designed for multiparty use without compromising the provenance privacy of the senders. ORAM-based (Oblivious RAM) techniques enhance data security by enabling encrypted datasets; however, they cannot handle a high number of servers and multiple clients without a non-collusion assumption and typically require a private state for each client interacting with the ORAM server.

Fully Homomorphic Encryption (FHE) is a widely used building block for constructing PSI protocols due to its single-round communication requirement. However, PSI protocols based on FHE,[21, 23, 38], face several challenges in addressing the PSMT problem, including key management, efficiency, and security issues. Theoretically, given a set held by the sender X and receiver query y, existing PSI protocols homomorphically compute a *sender polynomial*, $f(y) = r\prod_{x\in X}(x-y)$, where r is a random mask. To apply this method for PSMT, each sender would individually calculate their own sender polynomial, after which the *multiplicative* aggregation of these polynomials would correctly be an encryption of zero for an intersection and a random nonzero value otherwise. However, it would require FHE parameters that can tolerate $O(log(l))$ *additional* multiplicative depth for even a moderately large number of parties l (e.g., $l = 64$). Sender partitioning, although reducing multiplicative depth during polynomial computation, increases depth during result aggregation, particularly with multiple sender polynomials resulting from multiple senders. Moreover, using solely FHE to solve PMST would require non-collusion among all the parties, and a malicious receiver holding FHE secret keys would be able to monitor and decrypt all communications to/from the senders. Threshold-FHE can be used to handle such issues, where secret key shares are distributed to the parties. It facilitates better key management, prevents unauthorized decryption, and avoids a single point of failure if one of the parties acts adversarially.

Our PSMT protocol handles encrypted input for both the receiver's element and sets held by the sender and does not rely on any preprocessing (besides encryption) of the sets beforehand, completely eliminating the need for the senders to access the sets in plaintext during query computation. Our protocol's construction is based on threshold-FHE, where the parties use cheap homomorphic additions to aggregate ciphertexts gathered from multiple sites. Using α-out-of-l threshold-FHE, our protocol can handle up to $\alpha - 1$ colluding parties where $\alpha < l/2$. The security of our protocol is derived from the post-quantum security of FHE [3], and it upholds the provenance privacy of the senders to the receiver. We provide further security countermeasures for adversaries in the IND-CPA^D model by using existing noise-smudging techniques. We assume a semi-honest model where all parties operate on homomorphically encrypted datasets. In summary, we construct a protocol that can tolerate a large number of senders and efficiently compute on encrypted sender sets that may require frequent set updates.

The contributions of this work are summarized as follows:

- We define the Private Segmented Membership Test (PSMT) problem, which is widely relevant to real-world scenarios. Existing approaches result in various limitations, and we present a novel solution to address them.
- We address the shortcomings of existing PSI and MPSI-based approaches using finite-field FHE for solving PMST, which results from encrypted user data segmented across many senders and high aggregation latency. For the first time, we provide a novel summation-based set intersection protocol with approximate arithmetic threshold-FHE that overcomes these limitations and handles collusion among parties under an honest majority assumption, providing IND-CPA^D security.
- For the technical challenges in solving the PSMT problem with our novel solution, including plaintext domain size, function approximation accuracy/latency, throughput, and parameterization, we provide concrete parameters and novel strategies to deal with such issues and achieve good performance even for a very large number of senders and set sizes.
- We implement our method and present an experimental evaluation of our solution to show its significant performance advantage in the case of a large number of senders. Our anonymized source code is available for reproducibility and future research at https://anonymous.4open.science/r/psmt-7777. We show up to 2.4× to 5.6× performance improvement over previous works.

2 RELATED WORK

2.1 Private Set Intersection (PSI)

The first PSI protocol was based on the Diffie-Hellmann (DH) key agreement scheme [78]. This protocol leveraged the commutative properties of the DH function and offered security against the random oracle model. Its low communication cost continues to serve as a foundation for many modern PSIs. Freedman *et al.* [49] introduced PSI protocols based on oblivious polynomial evaluation (OPE) where sets are represented as polynomials. Additionally, PSI protocols have been constructed using Oblivious Pseudo-Random Functions (OPRFs) [48, 63], garbled circuits [90, 110], oblivious transfer (OT), and OT-extension [46, 70, 88, 94, 98]. Recent PSI protocols for unbalanced set sizes use OPE and increasingly utilize FHE with post-quantum security [21, 23, 38, 62, 102].

The two-party Private Set Intersection (PSI) model is extensively studied due to its wide real-world applications. Several variants of this model exist, where either both parties learn the intersection (mutual PSI) [44, 59] or only one of the parties learns the intersection (one-way PSI) [23, 92]. Other variants also allow the computation of different functions on the intersection [36, 44, 59, 60, 111]. Many of these protocols scale to millions of items within seconds and are only slightly slower than the simple but insecure method that exchanges hashed items. Pinkas *et al.* [94] used (1-out-of-n) OT based on [69]. The limitation of their approach is that OT step requires the sender to access elements in the hash table's bins, and extending it to substantial parties requires multiple Oblivious Pseudo-Random Function (OPRF) evaluations via OT, greatly increasing communication overhead.

CLR17 [23] protocol, and its improved variants [21, 38] are state-of-the-art FHE-based PSI protocols to the best of our knowledge. The basic protocol in [23] has the sender sample a uniformly random non-zero element r_i and homomorphically compute the intersection polynomial $z_i = r_i \prod_{x \in X}(c_i - x)$ using encrypted receiver's set $(c_1, c_2, \ldots, c_n)$ and the sender's unencrypted elements $x \in X$. z_i is returned to the receiver, who concludes that $y \in X$ iff $z_i = 0$. The receiver only learns the presence of an intersection. The CLR17 protocol applies many optimizations, including receiver and sender side batching using cuckoo hashing and binning, SIMD (Single Instruction Multiple Data) using FHE, and windowing. Later works, [21, 38] used an OPRF preprocessing on the encoded sender set, achieved malicious security, applied the Paterson-Stockmeyer algorithm for evaluating the intersection circuit, and reduced the communication by using extremal postage-stamp bases. These protocols require the sender to access the set in plaintext for the aforementioned optimizations and encodings for creating the polynomial for interpolation. Consequently, the privacy of datasets held by the sender is only protected against the receiver and not against the sender. Fundamentally, the CLR17-based protocols perform PSI by employing zero as a *multiplicative annihilator* in the polynomial $\prod_{x \in X}(y - x)$. Adapting these methods to the multi-party scenario would drastically increase the multiplicative depth required to obtain the query result and result in scalability issues.

2.2 Multi-party PSI (MPSI)

Multi-party PSI (MPSI) extends the two-party PSI problem to scenarios involving more than two parties. Two-party PSI protocols can be extended to multiple parties to handle the MPSI scenario; however, these solutions often lead to privacy and performance issues [108]. Several techniques have been employed to design MPSI, such as circuit-based computations [91], bloom filters [9, 80, 81], OPE [27, 49, 67], and OT and permutation-based hashing [89]. Kolesnikov *et al.* [71] used a technique based on oblivious evaluation of a programmable pseudorandom function (OPPRF) to implement a time-efficient MPSI protocol for large amounts of items. However, their time complexity scales quadratically w.r.t the number of parties in the protocol. Chandran *et al.* [18] improves upon [71] in terms of communication and extends it to circuit-based PSI and quorum-PSI. Notably, these protocols provide intersection results to all or some parties based on the intersection outcome and do not support a substantial number of parties.

Badrinarayanan *et al.* [7] employ threshold FHE to construct threshold MPSIs with sublinear communication complexities with thresholds proportional to the number of elements in datasets. They use a similar polynomial encoding of each set element as in [23]. Bay *et al.* [8] provide two MPSI protocols based on bloom filters and threshold homomorphic public-key techniques. Their protocol performs better than previous state-of-the-art [71] in terms of run time, given that the sets are small and a large number of senders exist. Nevo *et al.* [84] construct concretely efficient malicious MPSI protocols based on the recently introduced primitives such as OPPRF and oblivious key-value store (OKVS).

Table 1: Comparison of existing works to our work. Notation: l parties; α parties are corrupted and colluding; S.A.S: Security against the senders; L.F.P.: Low false positive rate (less than 10^{-3}); Agg. Comp.: Multiplicative aggregation overhead of FHE; OPRF, OPPRF, DFP, and HE denote Oblivious Pseudorandom Function, Oblivious Programmable Pseudorandom Function, Distributed Point Function, and Homomorphic Encryption, respectively. Post-Quant. refers to the security against quantum computer-capable adversaries; − denotes not applicable.

Protocol	Construction	Class	Post-Quant.	S.A.S	L.F.P.	Agg. Comp.	Collusion	Rounds	Adversary Model
Chen *et al.* [23]	FHE	PSI	✓	×	✓	$O(\log l)$	−	2	Semi-honest
Chen *et al.* [21]	FHE, OPRF	PSI	×	×	✓	$O(\log l)$	−	2	Malicious
Cong *et al.* [38]	FHE, OPRF	PSI	×	×	✓	$O(\log l)$	−	2	Malicious
Kolesnikov *et al.* [71]	OPPRF	MPSI	×	×	✓	−	$\alpha < l$	4	Semi-honest
Ramezanian *et al.* [97]	Bloom/Cuckoo Filter, HE	PMT	×	×	×	−	−	2	Semi-honest
Pinkas *et al.* [94]	Oblivious Transfer	PSI	×	×	✓	−	−	2	Semi-honest
Bay *et al.* [8]	Bloom Filter	MPSI	×	×	×	−	$\alpha < l$	5	Semi-honest
Nevo *et al.* [84]	OPPRF, OKVS	MPSI	×	×	✓	−	$\alpha < l$	4	Malicious
Vadapalli *et al.* [104]	DPF	ORAM	×	✓	✓	−	−	$\log(l) + 1$	Semi-honest
This work	Threshold FHE	PSMT	✓	✓	✓	$O(1)$	$\alpha < l/2$	4	Semi-honest

2.3 Other Similar Methods

Private Membership Test (PMT). PMT, or Private Set Inclusion, is a similar problem to PSI, in which a receiver learns if their single element is included in a sender's database without revealing anything to the sender. To solve PMT, many works apply Private Information Retrieval (PIR) based protocols that allow a user to retrieve an item from a database without the database owner learning anything about the item [35, 51, 73, 83]. PSMT closely matches the PIR; however, the sender's database is public in PIR. PMT has been extensively studied, particularly for two-party PSI in malware detection [97]. While hashing seems a naive solution for low-latency multi-party PMT, it becomes insecure with low-entropy input domains, and even high-entropy input domains, it may leak repeated elements upon consecutive executions. One can solve PSMT using individual PMT protocols with all senders via 1-out-of-n OT-based PMTs, followed by a secure XOR computation by the client. However, this approach has several drawbacks. Firstly, it necessitates sender access to plaintext sets for OT, losing privacy. Secondly, it requires the client to run n PMT protocols with n senders, adding extra communication and computation. While OT extension-based protocols can reduce communication, they also demand access to plaintext sets, and any updates in databases result in significant performance and communication penalties.

Some works have applied the PIR protocols to the PMT problem [33, 97] based on homomorphic encryption and bloom filters, but they induce significantly high false positives. Kulshrestha *et al.* [72], and Wang *et al.* [106] have constructed PMT protocols for identifying harmful media content and to detect password reuse across multiple websites, respectively. Tamrakar *et al.* [103] propose a carousel method for PMT for solving malware detection based on Trusted Execution Environments (TEEs). However, TEEs suffer from side-channel attacks and hardware-based attacks [20, 54, 77, 105], which have decreased their confidence for use recently.

Oblivious RAM (ORAM). ORAM allows a client to outsource storage of data to a server, and enable read and write operations to that data without revealing anything to the server about the data [53]. Many state-of-the-art ORAM frameworks [57, 58] require only constant client bandwidth blowup and low client storage but rely on weaker non-cryptographic security assumptions. ORAM-based techniques can be employed to solve PSMT but only partially. Namely, it is primarily designed for a single private database that can only be accessed by a single client, and using multiple servers requires a strong non-collusion assumption between them [57]. Distributed ORAM (DORAM) is a variant that handles multiple non-colluding servers and can be used for PSMT, but data is duplicated across the servers in DORAM. Furthermore, DORAM has higher bandwidth requirements and induces significant overheads while scaling for a higher number of senders. DUORAM [104] is one of the state-of-the-art DORAM models, however, it provides instantiations for up to only 3-party computation, which is far less than the scale involved in our scenario. Although the database on the server is encrypted in ORAM (similar to PSMT), ORAM requires the client to have a private state, due to which multiple clients cannot interact with the ORAM server directly, and more importantly, any querier who doesn't have access to this state cannot interact with the server.

In summary, PSMT can be addressed using general PMT-solving methods based on PIR, OT, ORAM, or TEEs, but they only offer partial solutions. The considerable overhead for a large number of senders, a lack of privacy for datasets held by senders, either from the client or the sender(s), along with high latency and low throughput typically associated with PMT protocols [103], render them impractical for efficient PSMT solutions. We compare representative works to our work in Table 1.

3 PRELIMINARIES & DEFINITIONS

In this section, we summarize some of the important notations for FHE and PSMT. We provide a complete list of notations in Table 2.

3.1 Fully Homomorphic Encryption

Fully Homomorphic Encryption (FHE) is a cryptographic primitive that allows computation on encrypted data with post-quantum security. Noise associated with an FHE ciphertext grows corresponding to each homomorphic operation, i.e., additively with additions and multiplicatively with multiplications. The most prominent FHE schemes are BGV [17], B/FV [15, 47], CKKS [29], and TFHE [34]. In practice, FHE schemes are often implemented as Somewhat Homomorphic Encryption (SHE) schemes where the user(s) provide the multiplicative depth required by the computation at the setup phase. In this work, we use the CKKS scheme, which uses a fixed-point complex number encoding to enable homomorphic computations on real numbers. Similar to B/FV and BGV, CKKS has operands in $\mathcal{R} = \mathbb{Z}[X]/\langle\Phi_M(X)\rangle$, where $\Phi_M(X)$ is the cyclotomic polynomial $(x^N + 1)$ of order $M = 2N$ (cyclotomic index) and degree $N \in \mathbb{Z}$ which is the ring dimension. CKKS parameters include the ring

Table 2: List of notations and descriptions

Notation	Description
l	The total number of parties ($l-1$ senders)
X_i	Set of the i^{th} sender (owned by the data-owner)
X_{l-1}	Senders' leader
$\mathcal{X}$	Union of the senders' sets
δ	The length of the bit-string
c_{x_i}	Ciphertext of the i^{th} sender
c_y	Ciphertext of the receiver
N	The ring dimension in FHE (power of 2)
q	Ciphertext modulus
D	The FHE multiplicative depth
η	The batch size of FHE scheme
α	Number of secret-key shares
σ	Standard deviation of smudging noise
a	Number of adversarial queries
s	Statistical security bits
λ	Computational security bits
$D_{R,\sigma}$	Discrete Gaussian noise distribution
L	Lower bound of interval for DEP
R	Upper bound of interval for DEP
n	Number of iterations for DEP
c	Degree of polynomial for Chebyshev approximation
j	Count of 1^{st} homomorphic square operation
k	Count of 2^{nd} homomorphic square operation
ρ	Scaling factor for reducing false positives
τ	Threshold required to confirm an intersection
κ	Limit for the random mask
$diff_i$	FHE-ciphertext (vector denoting $c_{x_i} - c_y$)
$etan_i()$	A piecewise function that takes $diff_i$ as input
K	Output of $etan_i()$ when input is 0 (maxima of VAF)
S	Parameter for controlling input range of 0 in VAF

dimension, ciphertext modulus, and standard deviation of the error. We employ the CKKS parameters to maintain 128-bit security in both classical and quantum contexts [4, 74, 79].

SIMD: In FHE, we can consider the factorization of (x^N+1) modulo p where p is a prime. We can then write the message space as a direct product of small fields, encrypt a vector of elements of these fields, and operate in parallel on the entries of these vectors, thus obtaining single instruction, multiple data (SIMD) capabilities [101].

In general, we have $(x^N+1) = f_1(X). \cdots .f_\eta(X) \pmod p$ with all f_i's having the same degree d such that $N = \eta \cdot d$ and message space is $\mathbb{Z}_p[X]/\langle x^N+1\rangle = \prod_{i=1}^{\eta}(\mathbb{Z}_p[X]/\langle f_i(X)\rangle) = (\mathbb{F}_{p^d})^\eta$. The plaintext space is isomorphic to η copies of the finite field with p^d elements, and instead of encrypting one single high-degree polynomial, we can encrypt a vector of η elements of $\mathbb{F}_{p^d}$. Therefore, a single homomorphic operation can handle η messages, each stored in a ciphertext slot, with the total slots and batch size equals η.

3.2 Threshold FHE

For the threshold functionality in our protocol, we utilize α-out-of-l (leveled) threshold-FHE (thresFHE) where l is the number of parties and α is the minimum number of partial decryptions needed to complete the decryption [12]. A thresFHE scheme consists of a tuple of probabilistic polynomial time (PPT) algorithms (*ThresFHE.Enc*, *ThresFHE.Eval*, *ThresFHE.PartialDec*), and two l-party protocols (*ThresFHE.KeyGen*, *ThresFHE.Combine*) with the following functionalities:

- *ThresFHE.KeyGen*$(1^\lambda, 1^D, params) \rightarrow (pk, evk, \{sk_i\}_{i\in[n]})$: Given a security parameter λ and a depth D, each party P_i outputs a common public key pk for encryption, a common evaluation key evk, and a secret key share sk_i of the implicitly defined secret key sk under some public parameter $params$.
- *ThresFHE.Enc*$(pk, m) \rightarrow c$: Given a public key pk, a message m, the encryption algorithm uses error distributions χ_{enc} and χ_{err} to sample $u \leftarrow \chi_{enc}$ and $e_0, e_1 \leftarrow \chi_{err}$ and outputs $c \leftarrow u \cdot pk + (m + e_0, e_1) \bmod q$ such that, $c = (c_0, c_1)$ where the ciphertext space is defined as $\mathcal{R}_q^2 = (\mathcal{R}/\langle q\rangle)^2$.
- *ThresFHE.Eval*$(evk, f, \{ck_i\}_{i\in[v]}) \rightarrow c^*$: Given an evaluation key evk, a v-input function, f that can be evaluated using at most depth D and ciphertexts c_i, the evaluation algorithm outputs a new ciphertext c^* that is an encryption of $f(m_1, \ldots, m_v)$. where $c_i \leftarrow$ *ThresFHE.Enc*(pk ,m_i).
- *ThresFHE.PartialDec*$(c, sk_i, \chi_{smg}(B_{smg}))$: Given a ciphertext $c = (c_0, c_1)$, a secret key share sk_i and a smudging error distribution $\chi_{smg}(B_{smg})$ with a bound B_{smg}, the partial decryption algorithm samples a smudging error $e_i^{smg} \leftarrow \chi_{smg}(B_{smg})$, and computes $pdec_i \leftarrow c_1 \cdot s_i + e_i^{smg}$.
- *ThresFHE.Combine*$(pk, \{pdec_i\}_{i\in[I]}) \rightarrow m$ or $\perp$: Given a public key pk, a set of partial decryptions $\{pdec_i\}_{i\in[I]}$ for an index set $I \subseteq [n]$ the combine algorithm computes $c_0 + \sum_{i=0}^{I} p_i \bmod q$ and outputs m if $|I| \geq \alpha$ otherwise $\perp$.

The key generation phase in thresFHE can be accomplished using a trusted setup procedure which can be run either via trusted hardware or secure multiparty computation to broadcast partial secret key to α key-holders. Existing works [6, 65] have shown that the latter method of key generation can be completed in a two-round, l-party protocol to compute a common public key, a common public evaluation key and a private share of the implicitly defined secret key. Similarly, thresFHE final decryption is a one-round α party protocol. As in standard homomorphic encryption schemes, we require that a thresFHE scheme satisfies compactness, correctness, and security [7].

3.3 Private Segmented Membership Test (PSMT)

Problem Definition: The PSMT problem is described in Figure 2. For a party P_y, with a data element y, and $l-1$ parties $P_1, P_2, \ldots, P_{l-1}$, each with a set X_i such that $\mathcal{X} = \sum_{i=1}^{l-1} X_i$, a PSMT allows the party P_y to learn $\{y\} \cap \mathcal{X}$, without leaking any elements not in any X_i or which party holds an element in the intersection. The sets held by different senders in the PSMT are mutually exclusive or disjoint. The parties involved are referred to as the *sender* and the *receiver*. For strong protection of user data, each sender in PSMT, P_i does not have plaintext access to X_i, i.e., each sender only has encryptions of $x \in X_i$. PSMT outputs $\{y\} \cap \bigcup_{i=1}^{l-1} X_i$ to the receiver and nothing to the senders without leaking any other information about the receiver's element and sets held by the senders.

Threat Model: PSMT protects the outsourced data owned by the data owners from the semi-honest senders, and our threat model is similar to that in previous works [10, 95]. The provenance privacy of the senders is preserved in PSMT. This ensures that in the

Parameters: PSMT involves l entities, namely $P_1, P_2, \ldots, P_{l-1}$ and P_y where, P_y is the receiver, P_{l-1} is the senders' leader and the rest of the parties are the senders. All senders possess encrypted sets of items with a bit-length of δ. The receiver holds an element y with a bit-length of δ; senders $P_1, \ldots, P_{l-1}$ hold encryption of sets $X_1, \ldots, X_{l-1}$, namely $c_{x_1}, \ldots, c_{x_{l-1}}$ and we define $\mathcal{X} = \bigcup_{i=1}^{l-1} X_i$.
Input: Encryption of y and encryptions of the sets $X_1, \ldots, X_{l-1}$.
Output: Receiver gets $\{y\} \cap \mathcal{X}$, and the senders, including the sender's leader, get $\perp$.

Figure 2: Ideal functionality of PSMT

event where $y \in X_i$, the set X_i is hidden from the receiver, who only learns that y is in some sender's set and not any particular set. We do not consider membership inference attacks with repeated adaptive queries. Such attacks can be mitigated by rate limiting or OPRFs [21] at the cost of additional overhead from preprocessing. Since parties are semi-honest, they are guaranteed to use the actual inputs; therefore, there is no integrity tampering [56]. We also do not consider integrity attacks where an adversary may homomorphically modify the sender's set. Securely storing hashes of ciphertexts (e.g., in an isolated storage) can prevent this.
Adversary Model: We assume a semi-honest setting, where parties (both senders and receiver) are assumed to be honest but curious. For the threshold functionality, we employ α-out-of-l thresFHE that can handle $\alpha - 1$ colluding participants where $\alpha < l/2$ such that the majority of the participants are honest. Adversaries will try to learn any of the receiver query y (if the receiver is not compromised) and the sender's set elements $x \in X_i$. Adversaries may eavesdrop on messages or compromise up to $\alpha - 1$ parties.

We note that in the rest of this paper, senders' sets refer to the encrypted sets of data owners that are held by the senders.

4 PSMT PROTOCOL

4.1 Basic Protocol without Approximation

Protocols using FHE to implement PSI [21, 23, 38] use zero as a multiplicative annihilator. Multiplications are not scalable, so we propose the novel idea of using additive aggregation.

We consider set elements to be members of $\mathbb{Z}$ (e.g., hashed elements). We describe our basic protocol in Figure 3 as a strawman protocol. The receiver has access to a query element y, and s/he creates replicas of y such that the number of replicas equals η (batch-size) and encrypts them to obtain c_y such that all slots of c_y contain y. The receiver then sends c_y to l senders. Senders either encrypt their sets themselves or receive the encryptions from data owners through a secure communication channel. Thus, each sender $i \in [1, l-1]$ has access to an encryption of a set X_i, c_{x_i}. Each slot of c_{x_i} contains different elements of X_i. Due to batching, each sender possesses only a single ciphertext. In case $|X_i| > \eta$, the sender possesses multiple c_{x_i}. Each sender i then computes a homomorphic difference of c_y, and its ciphertext(s) c_{x_i} in a SIMD fashion to obtain diff_i. Each sender i then computes a piecewise function $etan_i()$, defined below.

$$etan_i(\mathit{diff}_i) = \begin{cases} K, & \text{if } \mathit{diff}_i = 0 \\ 0, & \text{if } \mathit{diff}_i \neq 0 \end{cases}$$

Input : Receiver provides a query element y; senders possess encryption of sets $X_1, \ldots, X_{l-1}$, namely $c_{x_1}, \ldots, c_{x_{l-1}}$ such that $\mathcal{X} = \bigcup_{i=1}^{l-1} X_i$. Each set contains bit strings of length δ. K is the output of the function $etan()$ when input is zero. δ, and K are public parameters.
Output: Receiver outputs $\{y\} \cap \mathcal{X}$; senders outputs $\perp$.

1. **Setup**: The senders, the receiver, and the data owners jointly agree on a public-key thresFHE scheme with the same ring dimension. Receiver, senders, and data owners are provided an evaluation key evk and a public key pk. A secure multi-party computation or trusted hardware is used to distribute α partial decryption keys $sk_1, sk_2, \ldots, sk_\alpha$ among receiver and $\alpha - 1$ senders such that $\alpha < l/2$.
2. **Encryption**: The receiver encrypts its element y such that all slots of the resulting ciphertext c_y contain replicas of y and sends c_y to all of the senders. Senders encrypt each of sets $X_1, \ldots, X_{l-1}$ held by them using pk. Alternatively, data owners perform the encryption of the sets and send it such that, $c_{x_1}, \ldots, c_{x_{l-1}}$ ciphertexts are possessed by senders. If $|X_i| > \eta$, then sender i possesses multiple ciphertexts.
3. **Intersection**: Using c_y(s) each sender i:
   a. Homomorphically computes $\mathit{diff}_i = c_y - c_{x_i}$ and $etan_i(\mathit{diff}_i)$ using evk. If possessing multiple ciphertexts, sender computes multiple diff_i and $etan_i(\mathit{diff}_i)$ with the rest of ciphertexts parallelly and homomorphically summates them to a single $etan_i$.
   b. Sends $etan_i$ to a leader sender, who uses evk to homomorphically compute, $z = \sum_{i=1}^{l-1} etan_i$. Leader sender returns the ciphertext z to the receiver and $\alpha - 1$ senders for partial decryption.
4. **Partial Decryption**: Each sender i such that $i \in [1, \alpha - 1]$ partially decrypts z using sk_i and sends their partial decryption to the receiver. The receiver partially decrypts their share of z.
5. **Result interpretation**: The receiver combines α partial decryptions to decrypt z and outputs

$$\text{Result} = \begin{cases} \text{Included} & z = K \\ \text{Not Included} & z \neq K \end{cases}$$

Figure 3: Basic PSMT protocol

$etan_i()$ maps the input to K if the input is zero and zero otherwise. If equipped with multiple diff_i, senders compute separate $etan_i()$ for every diff_i in parallel since they are independent. The sender then simply sums the $etan_i$ into a single ciphertext. The results obtained by the senders can be additively aggregated by computing $\sum_{i=1}^{l-1} etan_i$, whose value is non-zero for an intersection and zero for a non-intersection. If the senders' sets are disjoint, the summation value will be K for an intersection; otherwise, it will be some multiple of K. We note that in practical deployments, it is most probable that some leader sender will take on the additional burden of aggregation or that different senders will take turns doing this. In our protocol, a leader sender aggregates the results.

This basic protocol can be implemented with any FHE scheme, and the aggregation would be composed of efficient homomorphic additions only. Unfortunately, this version would suffer from high multiplicative depth when computing the condition "if $\mathit{diff}_i = 0$." This leads to the need to approximate the function $etan_i()$.

4.2 Novel Value Annihilating Function (VAF)

One naïve thought is to approximate the function $etan_i()$ with a polynomial, but doing so would require polynomials with high degrees. We begin by considering the computation a single sender must perform. Suppose there exists a small negligible value ϵ and

Parameters: X_i is the input set whose encryptions are provided to sender i such that $i \in [1, l-1]$, y is the receiver's input. τ is the threshold value, and κ is the limit for the random mask. λ denotes the computational security parameter, and set elements are of bit strings of length δ. λ, τ, κ and δ are public parameters. K and S are public VAF parameters. L, R and n are public DEP parameters. c is the degree of the polynomial used for Chebyshev.

1. [**Parameters**]
   a. **Threshold FHE**: Parties, including the receiver, and senders, agree on parameters $(N, q, \delta, D, \alpha)$ for the thresFHE CKKS scheme.
   b. **Key Distribution**: Parties run a secure multiparty computation protocol *ThresFHE.KeyGen* that provides each partial key shareholder $i \in [1, \alpha)$ a share of the secret key sk as one of $\{sk_1, sk_2, \ldots, sk_\alpha\}$ among the receiver and $(\alpha - 1)$ senders, and broadcasts the common public key pk and evaluation key evk. Alternatively, a trusted setup can compute the common encryption, partial, and evaluation keys. All the parties, including data owners, have access to the public encryption key pk and evaluation key evk after this step.
2. [**Encryption**]
   a. **Encrypt** X **& batch** : For all $x \in X_i$, such that $i \in [1, l-1]$, each sender i groups its values x into vectors in $\mathbb{R}^m$ of length m with elements in $\mathbb{R}$. Then, the senders i batches each vector into $2 \cdot m/N$ plaintexts and encrypts them using *ThresFHE.Enc*. As a result, each slot of the resulting ciphertext contains individual x values. Each sender encrypts and batches their sets independently. Alternatively, data owners can execute *ThresFHE.Enc* using pk and send encrypted sets to their corresponding senders i such that $i \in [1, l-1]$. Each sender i obtains a ciphertext c_{x_i} after this step. In case $|X_i| > \eta$, the sender can possess multiple ciphertexts encrypting X_i. Note that FHE batching is applied to either the receiver's or sender's sides, but not both.
   b. **Encrypt replicas of** y: Receiver constructs a vector of length η, with each element being $y \in \mathbb{R}$. Then, the receiver encodes the vector into a CKKS plaintext and encrypts it *ThresFHE.Enc*. As a result, each slot of the resulting ciphertext c_y contains y.
4. [**Compute Intersection**]
   Senders use *ThresFHE.Eval* with evaluation key evk to execute the following steps:
   a. **Homomorphically compute subtractions**: Each sender i receives the ciphertext c_y and computes $\mathit{diff}_i = c_y - c_{x_i}$.
   b. **Apply DEP to reduce the domain size**: Each sender i iteratively applies the domain extension process n times using DEP to shrink the domain interval of values diff_i from step 4a into the interval $[-R, R]$.
   c. **Homomorphically evaluate VAF**: Each sender i uses the Chebyshev approximation method to approximate $etan_i(\mathit{diff}_i) = K \cdot (1 - \tanh^2(S \cdot \mathit{diff}_i))$ in the interval $[-R, R]$ using a degree c polynomial.
   d. **First homomorphic squaring**: Each sender i homomorphically squares $etan_i$ and obtains $etan_i^{2^j}$ such that $j \in [1, 12]$. Higher values of $j \geq 2$ can be chosen depending on the desired approximation accuracy for larger senders' set sizes.
   e. **Scaling & second homomorphic squaring**: Each sender i multiplies $etan_i^{2^j}$ using a scaling factor, ρ. Step 4d can be applied again to obtain $(\rho \cdot etan_i^{2^j})^{2^k}$ with $k \in [3, 4]$ to exponentially increase the number of parties the protocol can handle and reduce the false positive rates to negligible or zero.
   f. **Homomorphically evaluate summation**: If a sender possesses multiple ciphertexts, Step 4a to 4e is applied to all the remaining ciphertexts in parallel, and the sender homomorphically summates them to $(\rho \cdot etan_i^{2^j})^{2^k}$. An aggregator collects $(\rho \cdot etan_i^{2^j})^{2^k}$ from all other senders, samples a random non-zero plaintext element $r \in [-\kappa, \kappa]$; and homomorphically evaluates z, such that,
   $$z = r + \sum_{i=1}^{l-1} (\rho \cdot etan_i^{2^j})^{2^k}$$
   z is then broadcast to the α parties holding the partial decryption keys, including $\alpha - 1$ senders and the receiver.
5. [**Partial and Final Decryption**]
   a. **Partial decryption & smudging noise** : Upon receiving z, each partial key-share holder sender $i \in [1, \alpha)$ uses their secret key share sk_i to partially decrypt z using *ThresFHE.PartialDec*, which introduces a smudging noise e_{smg} to the partial decryptions $part_i$. Each sender i computes $part_i$ and sends their decryption share to the receiver.
   b. **Final decryption**: The receiver uses their partial decryption key sk_i and z to execute *ThresFHE.PartialDec* and obtains their partial decryption share $part_i$. Optionally, the receiver can use a *ThresFHE.PartialDec* algorithm that does not introduce a smudging noise to increase the precision. Receiver then combines their share with the partial decryptions received from the $\alpha - 1$ senders, $\{part_i : sk_i\}_{i \in [0, \alpha)}$ using *ThresFHE.Combine* and pk and obtains the resulting vector of length η.
   c. **Interpretation of result**: The receiver checks if the magnitude of any element of the resulting vector exceeds τ,
   $$\{y\} \cap \bigcup_{i=1}^{l-1} X_i = \{y : \mathit{ThresFHE.Combine}(z) \geq \tau\}$$
   If any value in a ciphertext slot is greater than the threshold τ, an intersection exists; otherwise, no intersection exists.

Figure 4: Full PSMT protocol

a sender has access to a set X_i with size n such that $x_j \in X_i$ for $j \in [n]$ and is given an FHE ciphertext c_y encrypting a receiver's input $y \in Y$.

Consider the function $f(y, X_i) = \sum_{j=0}^{n-1} g(\mathit{diff}_j)$, where $\mathit{diff}_j = (y - x_j)$ and $g(\mathit{diff}_j) = \frac{1}{\mathit{diff}_j + \epsilon}$. Then, $f(y, X_i)$ can be thought to exhibit the following behavior similar to that of $etan_i$:

$$f(y, X_i) = \begin{cases} 1/\epsilon + \sum_{j=1}^{n-1} g(\mathit{diff}_j), & \text{if } \mathit{diff}_0 = 0 \\ \sum_{j=0}^{n-1} g(\mathit{diff}_j) \ll K, & \text{if } \mathit{diff}_j \neq 0 \end{cases}$$

Here, if $\mathit{diff}_0 = 0$ (i.e., $y \in X_i$) and $\epsilon \to 0$, $f(y, X_i)$ increases without bound and its limit approaches infinity. This misuses the notion of infinity in terms of computation, as infinity is not a numerical quantity. Nevertheless, it leads us to some intuition on how to construct a protocol that is amenable to PSMT: we should have a function that each sender computes that outputs very large values (close to K) to annihilate the summation on an intersection but outputs a negligible summation value compared to K on a non-intersection. Then, additively summing these results from each sender would yield a result indicating if the receiver's element y is in $\bigcup_{i=1}^{l-1} X_i$, i.e., the large value indicates intersections and small negligible values indicate non-intersections.

To realize this idea, we consider a candidate function $D_K^{Exact}(x) : \mathbb{R} \rightarrow \mathbb{R}$ that exhibits the useful properties described above without the problematic behavior of a potential division by zero: $D_K^{Exact}(x) = \begin{cases} K, & \text{if } x = 0 \\ 0, & \text{if } x \neq 0 \end{cases}$. This function $D_K^{Exact}()$ easily satisfies these constraints and has the exact same behavior as $etan_i()$. We call such functions *Value Annihilating Functions* (VAF). A VAF maps zero to K and all others to zero. Computing VAFs homomorphically is challenging because *if* conditions cannot be evaluated efficiently in FHE. We thus consider another function approximating $D_K^{Exact}(x)$. Instantiating a VAF on finite fields with exact schemes would require a very large plaintext domain for representing inputs, and the approximation procedure would be very inefficient. Hence, for such a function, approximate-arithmetic FHE is preferable.

It is challenging to find a function that accurately approximates such kind of behavior. The functions approximating such kind of behavior require computing the inverse or the division operation, and computing such an operation homomorphically is fundamentally challenging. Similarly, trying to control the maxima of such functions while approximation results in very bad accuracies. We explored many candidates for VAFs, such as rational functions including $f_{K,S}(x) = \frac{K}{(1+S\cdot x)^4}$, sigmoid, hyperbolic tangent, piecewise linear, sign function, among others. The rational functions require computing a division for VAF, which is inefficient for FHE. Sigmoid, hyperbolic tangent, piecewise linear, and sign functions either did not exhibit the kind of properties that we needed or resulted in a very high accuracy loss while approximating in FHE. With all the considerations, we discovered the following function exhibits satisfactory behavior for our purpose with minimal accuracy loss:

$$D_{K,S}^{Approx}(x) = K \cdot (1 - \tanh^2(S \cdot (x)))$$

This function is shown in Figure 5, approximating a VAF. Since piece-wise functions cannot be evaluated efficiently in FHE due to branching, we designed a function that gets larger quickly as the input gets closer to zero. The derivative of the hyperbolic tangent $(1 - \tanh^2(x))$ has maxima as one for zero inputs and approaches zero for inputs in the range $(-\infty,-3) \cup (3,\infty)$. We set K acts as the maxima and use S to increase the input range for zero (i.e., width of the function's hump near the zero) such that the function outputs zero for inputs in range $(-\infty,-1) \cup (1,\infty)$. Using this function for approximation, senders acquire a summation value that is equal to K when an intersection is detected; otherwise, they obtain a summation value that is close to zero.

When computing functions such as $D_{K,S}^{Approx}(x)$ in approximate homomorphic encryption, only additions and multiplications are available. However, such approximations are not zero-valued for all nonzero arguments or even all arguments outside some interval centered at zero. In the following sections, we describe how we compute a polynomial approximation $D_{K,S}^{Approx}(x)$ whose values on $\mathbb{R}$ except for a small interval near zero can be bounded by some small number κ. Then, we can set parameters K, S, κ, ν such that $(l-1)\cdot\kappa < \tau$, where $\tau = \nu \cdot K$. $\nu \in (0, 1]$ is a threshold proportion used to account for the case where error from approximate-arithmetic FHE may cause a sum of l outputs of $D_{K,S}^{Approx}(x)$ on nonzero arguments to be greater than $(l-1) \cdot \kappa$.

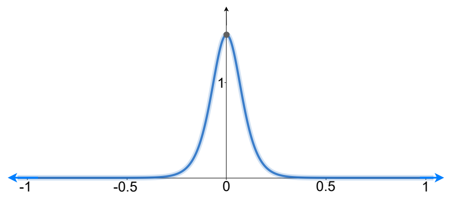


Figure 5: Graph of the VAF $K \cdot (1 - \tanh^2(S \cdot x))$: $K = 1.5$, $S = 10$.

4.3 Polynomial Approximation of VAF

The approximation of complex non-polynomial functions is a well-studied topic. Existing works such as [30, 31] use polynomial composition with iterative algorithms to approximate functions like min/max and comparison using FHE. Other works have used techniques like Taylor series [13, 66], minimax approximation [28, 100], look-up tables [40] and conversion between FHE schemes [10, 34].

A major challenge for approximation techniques is finding polynomial approximations that work on large domains. Existing approaches for approximation [13, 29, 64, 66] struggle for large domain intervals, inducing very high approximation errors. Even domains of thousands can be challenging [32, 64]. This would require homomorphically evaluating polynomials of an extremely large degree. For instance, simply using Chebyshev-based approximation for approximating non-linear functions for domains such as $[-1000, 1000]$, requires almost 20.5 minutes of computation time and an additional 12 multiplicative depth for acceptable accuracy. Hence, the error induced by the approximation and the computational cost of approximation are the major factors [32] to be considered during the approximation. For our application, we are primarily concerned with the computational cost of the evaluation and require low precision that will enable us to distinguish values converging to K (for intersection) and 0 (for non-intersection).

Domain Extension Polynomials (DEPs). DEPs enable the effective shrinking of a large domain interval $[-L^n R, L^n R]$ to a smaller subinterval $[-R, R]$ such that the property of the VAFs around zero in the smaller domain is preserved. To compress inputs from an interval $[-L^n R, L^n R]$ to an interval $[-R, R]$, we can iteratively apply a DEP with $O(n)$ operations and $2n$ additional depth. Using this method we can convert inputs $z \in [-L^n R, L^n R]$ into values $D(z)$ such that $z \in [-R, R] \implies D(z) \approx z$ and $z \notin [-R, R] \implies D(z) \approx sign(z)$. Specifically, we utilize Algorithm 1 from [32]. To handle inputs in $[-R, R]$ using the DEP $B(z) = z - \frac{4}{27}z^3$ at an iteration n of Algorithm 1, we divide inputs to $B(z)$ by $L^n R$, and scale its ouputs by $L^n R$. This converts $B(z)$ from a DEP on $[-1, 1]$ to a DEP on $[-R, R]$. If the accuracy of $B(z)$ is not good enough, then squaring its outputs can make $B(0)$ larger and $B(z)$ for $z \neq 0$ smaller, at the cost of additional depth and runtime. DEPs enable the approximation of values within a large domain interval, allowing the protocol to handle potentially millions of inputs efficiently.

Chebyshev Approximation. Chebyshev polynomial approximation are minimax-based polynomial approximation method that achieves the smallest possible polynomial degree with minimal approximation errors [32, 64]. Approximating functions using standard approximation techniques like Taylor series can induce the Runge phenomenon [14] that causes an approximation to yield poor accuracy at the edges of the interval. Chebyshev polynomials reduce this phenomenon and provide an approximation that is close to the best polynomial approximation to a continuous function.

We apply DEPs and Chebyshev approximation to approximate a $D^{Approx}_{K,S}(x)$ in a much smaller interval $[-R, R]$ while preserving the original larger domain $[-L^nR, L^nR]$ with low FHE multiplicative depth. This leads our protocol to approximate $etan_i()$ with low approximation errors and computational cost for a large domain.

4.4 Security

Using our basic protocol in Figure 3, the receiver can learn the aggregation value $\sum_{i=1}^{l-1} etan_i$ to infer information about the difference between their query and the sender's value to get information about the non-intersection values. To fix this issue, we require the senders to obscure the aggregation value by adding a small random masking term $r \in [-\kappa, \kappa]$, where κ is a public bound for the approximation value of VAF on a non-intersection.

Our protocol incorporates a threshold version of the CKKS scheme to withstand collusion among up to α parties where $\alpha < l/2$. This ensures that even if the receiver acts as an adversary, they cannot derive any additional information from the communication channels, as decryption is unachievable with only their single partial key. Consequently, the receiver would need to collude with $\alpha - 1$ parties to gain any extra information. Semantic security of our PSMT protocol follows directly from the security of the underlying threshold variant of the FHE scheme CKKS [29]. Recent works by Cheon *et al.* [25] and Li and Micciancio [75] have shown that IND-CPA security is not sufficient for both exact (BGV, BFV, and TFHE) and approximate (CKKS) FHE schemes. In particular, [75] presented a passive attack against CKKS and showed that CKKS is not secure if access to a decryption oracle is provided. They introduced a new notion of privacy called IND-CPA^D that provides access to encryption, evaluation, and decryption oracles. Moreover, the access to decryption oracle requirement is also present in thresFHE schemes for accessing the partial decryptions, and thus the threshold variants of all FHE schemes, including CKKS in our basic protocol, are exposed to IND-CPA^D attackers [19, 68].

Handling IND-CPA^D capable adversaries. In CKKS, the noise component is a part of the message, and this results in the linearity of the decryption function to the secret key revealing the decryption noise, and making it vulnerable to IND-CPA^D attackers [26]. To mitigate this kind of attack, existing works [19, 26, 42, 75], have proposed various countermeasures such as bootstrapping to reset the noise variance to a preset value, using the rounding procedure or attaching a proper noise at the end of the decryption process called smudging noise to avoid the linearity on the secret key. We employ the noise smudging (noise flooding) method to transform the thresFHE CKKS scheme achieving the weak IND-CPA security definition into one which is IND-CPA^D secure [76].

Before adding the noise, proper estimation needs to be done for noise addition using either static or dynamic noise estimation [5, 76]. In our application, we use static noise estimation since it can be performed offline to compute the noise using publicly available bounds on the inputs and the function to be evaluated. Senders estimate the noise using a fresh secret key-public key pair and select messages reflecting actual data for the homomorphic computation. The additional noise is drawn from $D_{R,\sigma}$ over the polynomial ring, represented in its coefficient form where σ is a standard deviation set by a security level and noise estimate. Thus, to achieve $s > 0$ bits of statistical security, one can set $\sigma = \sqrt{24a \cdot N} \cdot 2^{s/2}$ (see Corollary 2 in [76]), where a is the number of adversarial queries the application is expecting, and N is the ring dimension used for FHE. We add smudging noise using this distribution to ensure IND-CPA^D security for our underlying thresFHE CKKS scheme.

4.5 Choosing Appropriate Parameters

Over the course of our experiments, we encountered several challenges in determining suitable parameters for the DEP and Chebyshev approximation and tailoring them to the specific sender set sizes. To set up the DEP parameters for reducing the Chebyshev approximation degree, one needs to be careful while choosing L, R, and n with c to minimize FHE depth and approximation errors for a given $|\mathcal{X}|$. We provide further details regarding parameterization and list our findings in Appendix A and Table 5.

We note that the values described in this section and in Table 5 are useful for the case where $\bigcap_{i=1}^{l-1} X_i = \emptyset$, i.e., each sender's set has no element in common with other sender sets. In case senders' sets are not disjoint, we can allow up to Ψ of l senders to have a duplicate element, so long as values up to $\Psi \cdot K$ can be represented correctly in CKKS with the parameterization being used. To ensure that the summation does not overflow, we can set a smaller K and heuristically choose smaller values for parameters j, k, ρ, and τ. Note that in case, we allow duplicate items, our protocol will require higher precision bits to accommodate for the precision loss occurring during smudging noise addition for security.

Smudging Noise. In our protocol, we require the senders to add a smudging noise to their partial decryptions from $\sigma = \sqrt{24a \cdot N} \cdot 2^{s/2}$ that has a larger variance than the standard noise distribution of basic thresFHE CKKS. We set $a = 2^{10}$ and $s = 36$, limiting the adversary's success rate to 2^{-36} (about 1 in 68 billion) for 2^{10} adversarial queries for a single ciphertext. Li and Micciancio offer noise-estimation parameters for $a < 2^{15}$ only (see Table 1 in [76]) and having higher values of a without adding larger smudging noise or significant precision loss is a problem orthogonal to ours and a topic under research [19, 39, 76]. Hence, we assume that distributed thresFHE queries for identical or related ciphertexts are limited to fewer than 2^{10} in our application since all parties will need to be involved in every thresFHE decryption. One solution to mitigate such an issue would be refreshing the noise estimate after every 2^{10} queries which would add minimal overhead to the latency as it can be done offline. Similarly, another mitigation as suggested by [76], would be updating the secret key shares occasionally, however, this method will require extra rounds of communication.

After running the noise estimation, we estimated roughly 34 noise bits for smudging. This smudging noise ensures the IND-CPA^D security; however, it introduces noise with a high variance to each partial decryption. Moreover, the noise amplifies when aggregated across numerous parties, leading to a substantial loss in precision due to which non-intersection summation $z = \sum_{i=1}^{l-1} etan_i$ can overflow beyond negligible values. However, this is not an issue in our protocol since we can simply choose a higher τ value to accommodate for the precision loss. Moreover, the noise smudging is performed independently by the senders, which incurs very low latency and, thus, does not significantly affect the protocol's latency.

4.6 Asymptotic Complexity Analysis

Each sender must compute the procedures described in Figure 4. This requires $2n + O(log(c)) + j + k$ homomorphic multiplications. These procedures require a multiplicative depth of $2n$ for the application of the DEP, approximately $log(c) + 1$ for Chebyshev approximation, and $j + k$ for repeated squaring. The precise amount of depth recommended for a polynomial approximation of degree c is chosen heuristically[1]. CKKS operations besides multiplication, e.g., addition and scaling, contribute relatively small amounts of overhead and noise. The additive aggregation of each sender's result is relatively inexpensive; most notably, it does not increase multiplicative depth *even with an increasing number of senders.*

Our protocol requires sending one ciphertext from the receiver to the $l - 1$ senders and receiving $\alpha - 1$ ciphertexts from the senders by the receiver. Thus, the communication between the receiver and senders is bounded above by $l + \alpha - 2$ ciphertexts. The inter-sender communication is bounded by $(l - 1) \cdot \alpha$ ciphertexts which includes the final result ciphertext sent to the receiver by the aggregator. The number of communication rounds between the receiver and senders is four in our protocol, assuming a trusted setup provides all the parties their respective keys. In case a separate secure multi-party protocol is run for the key generation during the setup phase, it will add two more rounds to the overall communication.

4.7 Discussion

4.7.1 Set Updates. As we do not require any preprocessing of senders' sets, updates to the encrypted senders' sets are trivial in our protocol. Most computations in our protocol take place during the online phase of the protocol. This allows the protocol to easily add or delete any element in senders' sets, as no pre-processing is necessary during the offline phase. The data owners can simply update their sets, encrypt them, and send the corresponding ciphertext(s). Alternatively, data owners can only send the latest set of elements separately, which will save communication costs. Senders can then use all received ciphertexts to compute the PSMT for new queries. This will allow the parties to compute the intersection of their private sets on a regular basis with sets that are often updated.

4.7.2 Multiple Receivers. In case we have multiple receivers, each receiver must participate in the setup phase of key generation to obtain a partial decryption key. The threshold value α can be adjusted to accommodate the increased number of receivers. For r number of receivers, the threshold can be adjusted to $(\alpha - r)$ which will ensure that only the querying receiver needs to supply partial decryption to complete the decryption process. However, with multiple receivers, it's crucial they do not collude, as each holds a partial decryption key, and collusion can lead to fewer than $(\alpha - r)$ senders being required for decryption, as malicious receivers could get access to the final result by monitoring the communications channels. In scenarios where a new receiver wants to perform the query, we can update the secret key shares in thresFHE by updating the original polynomial used for setting up secret sharing.

4.7.3 Attacks on CKKS in OpenFHE. For our experiments, we utilize an open-source FHE library called OpenFHE [2]. OpenFHE uses non-worst-case noise estimation during static noise estimation to provide better efficiency [39, 41, 45, 87], and multiplies the noise internally to ensure enough noise bits suggested in [76]. Recently, Guo *et al.* [55] show that relying on non-worst-case noise estimation undermines noise-flooding countermeasures for achieving IND-CPA^D and implement a key-recovery attack on OpenFHE. In their attack, the adversary has the freedom to select a different evaluation function for noise estimation; however, in our application, the senders are semi-honest, who run the noise estimation, and add the correct amount of noise using worst-case noise estimation using [Table 1, [76]]. Hence, this attack does not apply to our protocol. Alexandru *et al.* [5] have also pointed out that these attacks are the result of misusing the OpenFHE library by employing different (incompatible) circuits during noise estimation.

4.7.4 Tradeoffs. The process of applying a DEP followed by Chebyshev approximation requires significant depth, but this depth does not depend on the number of senders involved. Each sender can perform this computation in parallel, but this step may be expensive. Thus, even though we require a higher computational complexity compared to other methods for a small number of senders, this complexity does not depend on the number of senders and stays moderately low for a significantly higher number of senders. The protocol's computation and FHE multiplicative depth only depend on the size of the sender's set and not the number of senders. We also note that FHE batching can only be applied to either the receiver or senders, but not both. Applying batching to the receiver's elements will require using hashing techniques (e.g., cuckoo hashing), which will eventually require pre-processing of the elements.

5 CHALLENGES

5.1 Increasing Throughput

Using our PSMT protocol, senders encode each of its sets' elements into separate plaintexts and individually compute the polynomial approximation. A single query can take time on the order of seconds, highlighting the need for better throughput. The throughput of our PSMT protocol can be improved using the SIMD feature available in FHE schemes. The CKKS scheme supports the packing of multiple message vectors into a single ciphertext where the slots of the ciphertexts hold different values. This allows slot-wise addition and multiplication [16, 50, 52, 101]. The CKKS scheme can encrypt $\eta = N/2$ elements in $\mathbb{R}$ into a single ciphertext using this mapping. The batching technique allows the sender to operate on η items simultaneously, resulting in η-fold improvement in both

[1]Some guidelines for choosing parameters for polynomial approximation can be found at https://github.com/openfheorg/openfhe-development/blob/main/src/pke/examples/FUNCTION_EVALUATION.md

the computation and communication. To enable batching when not all slots are used, we fill the remaining slots with a dummy value.

5.2 Supporting Larger Sets

Increasing the capacity of the protocol to support large senders' sets requires the use of larger plaintext spaces. In prior work on FHE-based PSI [21, 23, 38], the sender partitions their elements into disjoint sets and computes the intersection of receiver elements with each partition. This can reduce the effective set size in each computation, but partitioning increases the number of results to be sent back. Although the results can be multiplicatively aggregated, it requires more depth. An advantage of our design is that we gain the ability to partition computations and thereafter aggregate results from the partitions nearly for free.

Our protocol can be extended to use very large sender sets. However, we note that it will require more depth and computing power due to the large FHE parameters required for the DEP procedure and Chebyshev approximation. Since our final result is the summation of individual results from the senders, increasing the senders' set sizes does not dramatically increase the communication size.

5.3 Reducing False Positives

In the existing works, false positives occur due to hashing collisions [23] or representation of the set elements using some probabilistic encoding techniques like bloom filter encoding [99]. In our protocol, false positives can occur due to bad approximation accuracy of the VAF. After applying the DEP procedure, we found that the homomorphic approximation for the VAF was not accurate enough within the range $[-3, 3]$. While the approximation would correctly map zero values to K, the non-zero values in the range $[-3, 3]$ would be mapped to values far too close to K. Hence, instead of producing a hump at 0, we observed that the approximation would produce a flatter curve. These values would later contaminate the summation outcome, leading to false positives. To solve this issue, we used a technique involving homomorphic squaring and scaling. Homomorphically squaring the values in the range $(-1, 1)$ would map them to smaller and smaller values. Then, we would scale them by a scaling factor $\rho \in (2, 2.8)$ so they remain within $(-1, 1)$. Finally, we apply homomorphic squaring again, which would square ρ for intersection and augment the difference between values mapped to zero and non-zero. After using the aforementioned techniques, κ is the highest value approximated by the Chebyshev (see Table 5) when diff_i is at its minimum value of one.

Our complete protocol after incorporating the aforementioned solutions is detailed in Figure 4.

6 EVALUATION

We implemented our proposed PSMT protocol using C++17 and OpenFHE v1.0.3 [2]. The anonymized source code is available at https://anonymous.4open.science/r/psmt-7777. We used the threshold variant of CKKS scheme [29] and employed the default FHE parameters provided in OpenFHE using [3] and the noise smudging parameters provided in [76] to ensure a $(128, 36)$-bit computational and statistical security. To be able to support a higher amount of noise bits during noise smudging, we recompiled OpenFHE with NATIVE_SIZE = 128 flag. Our experiments were run on a server with an AMD EPYC 7313P processor and 128GB of memory, running Ubuntu 20.04. In our LAN setup, we assumed a 10 Gbps bandwidth and 0.2ms round trip time (RTT) latency, while our WAN assumed 200 Mbps and 1 Gbps bandwidths with an 80ms RTT latency.

We evaluate the performance in terms of computational and communication costs. For computational latency, we evaluate per-sender query runtime and the runtime for aggregating results from multiple senders by performing 5 runs for each combination of parameters and taking the average. The bit-length δ of the set elements is matched to the plaintext space accommodated by the DEP and Chebyshev parameters. The senders' individual computations are independent in our protocol. We assume that after receiving the query ciphertext from the receiver, each sender operates on their own set in parallel. We choose the number of senders for every senders' set size from 2^7 to 2^{25} such that each sender has to evaluate a single ciphertext, i.e., a single sender possesses at most η elements. In case we have fewer senders for a given $|\mathcal{X}|$, which results in a single sender possessing more than η elements, we assume that senders pack their set elements in multiple ciphertexts and evaluate them parallelly and homomorphically summates them to a single ciphertext. To obtain the noise for noise smudging, we assume senders perform static noise estimation offline using the publicly available input bounds reflecting actual data and the VAF function to be evaluated. This is a one-time estimation process that solely depends on the query computation time and induces negligible latency while considering PSMT for a large number of senders. To better understand the scalability of our protocol, we evaluate it on the range of the number parties $l = 2^n$ where $n \in [2, 12]$.

Baselines. Many variants of PSI, MPSI, and PMT protocols are designed to handle specific or more general scenarios. Selecting a specific protocol for comparison with our work is a complex task, given our novel privacy model with server-side encryption, provenance privacy, and a high number of senders. Considering the existing works that we discussed in Section 2, [71], [7], [97], [94] and [18] either lack scalability for a large number of senders, do not provide a public implementation or do not support multiple senders. ORAM-based approaches such as [104] simply do not support a higher number of senders for comparison with our work. [23] is an older work and serves as the foundation for [21] and [38]. Thus, we primarily compare our protocol to the following state-of-the-art baseline FHE-based and non-FHE-based PSI and MPSI protocols: Cong *et al.*, [38], Bay *et al.* [8] and Nevo *et al.* [84]. Cong *et al.*'s protocol uses the BFV FHE scheme [47] and for a fair comparison, we implemented their protocol for a multiparty setting using a threshold variant of BFV in OpenFHE by using their public implementation [1], originally implemented in SEAL [22]. We used the default noise-flooding mechanism in OpenFHE for BFV to ensure IND-CPA^D in the threshold setting. Similarly, for non-FHE-based protocols we ran the public implementations (both in C++) of the MPSI protocols of Bay *et al.* [8], which is one of the fastest-known MPSI protocols for smaller set sizes and a large number of parties and Nevo *et al.* [84], which is one of the most scalable maliciously secure MPSI protocol based on OPPRF and OKVS on our server for performance comparison.

Table 3 shows the overall communication overhead and computation time, which is the total time required to complete the DEP and

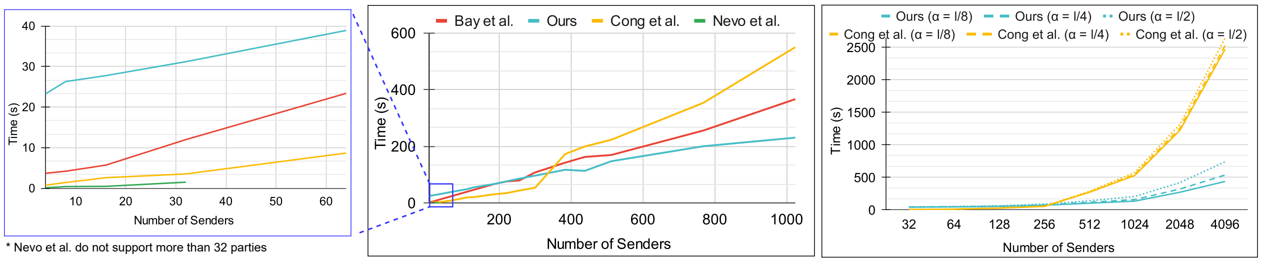


Figure 6: Runtime comparison for Bay *et al.* [8] and Nevo *et al.* [84] with ours and Cong *et al.* [38] (middle, zoomed-left). Total computation time comparison for Cong *et al.* [38] vs. our protocol for $l < 4096$ and $\alpha \in \{\frac{l}{8}, \frac{l}{4}, \frac{l}{2}\}$; senders' set size is set to 2^{15} (right).

Chebyshev approximation using the parameters in Table 5. Observe that the size of messages from the receiver to senders increases with senders' set sizes due to larger senders' set size requiring query ciphertexts, which should be able to tolerate a higher amount of noise and, hence, higher FHE depth for the DEP and Chebyshev. The inter-sender communication is largely determined by the size of ciphertexts sent by senders to the leader after query computation. The leader sender has to then send back α aggregated ciphertexts for partial decryption. Since the aggregated ciphertext's size is significantly smaller, the communication is not affected much by higher values of α in our protocol (Figure 6, right). Finally, the message size from the senders to the receiver depends on α, and it remains very low as it only contains the summation result ciphertext.

For various settings of $|\mathcal{X}|$, we set the highest number of parties to 1024 (modest for most applications discussed in Section 1) and examine the aggregation latency and communication overhead. Most related state-of-the-art works are only evaluated for a small number of senders (usually 100 or less) [8, 9, 71]. The aggregation time (not counting encryption and partial/final decryption) of our protocol stays within 100 seconds for $|\mathcal{X}| \leq 2^{25}$, and our protocol can be easily extended for a higher number of senders (up to 4096 in Figures 6 and 8). Moreover, the corruption threshold α does not significantly affect the computational or communication overhead. For the applications discussed in Section 1, we can observe that our protocol can handle the increasing number of parties very efficiently. The parties do not need to store any auxiliary data for computing PSMT. Moreover, parties in these applications (e.g., government agencies, banks, insurance companies) are often equipped with high bandwidth networks and connected through LANs, which helps minimize the communication latency in our protocol. We note that we only report computational latency for only up to 4096 parties, but this is only due to memory constraints when running the protocol with a huge number of senders in a single system.

Comparison with FHE-based protocol. We compare the computational latency of our protocol and Cong *et al.*'s protocol [38] in the multi-party setting in Figures 6 to 8. For the number of senders less than about 320, [38] is faster than our protocol, but for higher number of senders, our protocol outperforms [38]. As the number of senders increases, aggregation time for [38] becomes the bottleneck due to *multiplicative* aggregation. The query computation time for our protocol is slower than [38]. Similarly, partial decryption time and ciphertext combining time are also slower for our protocol, as these operations are inherently slower in CKKS compared to BFV. However, our protocol is almost 3.3 – 5.6× faster for various values of α in Figure 6 (right), and $|\mathcal{X}|$ in Figure 7 when the aggregation

Table 3: Communication and computation overhead in our protocol. Agg. denotes the total time required to aggregate the ciphertexts. R-to-S, S-to-S, and S-to-R denotes receiver to sender, inter-sender, and sender to receiver, respectively. Query refers to the query computation time for each sender.

$\lvert\mathcal{X}\rvert$	Message size (MB)			No. of	Agg.	Query	Depth
	R-to-S	S-to-S	S-to-R	Senders	(second)	(second)	
2^7	45	49.1	2.1	128	6.53	13.20	21
2^8	49	53.1	2.1	256	14.12	15.24	23
2^{10}	63	67.1	2.1	1024	40.29	19.09	30
2^{13}	79	83.1	2.1	1024	67.51	26.00	38
2^{15}	87	91.1	2.1	1024	71.16	30.07	42
2^{20}	107	111.1	2.1	1024	87.78	59.67	52
2^{21}	111	115.1	2.1	1024	103.98	64.65	54
2^{22}	115	119.1	2.1	1024	84.63	68.90	56
2^{23}	119	123.1	2.1	1024	90.34	75.80	58
2^{24}	123	127.1	2.1	1024	84.09	81.08	60
2^{25}	127	131.1	2.1	1024	92.53	86.76	62

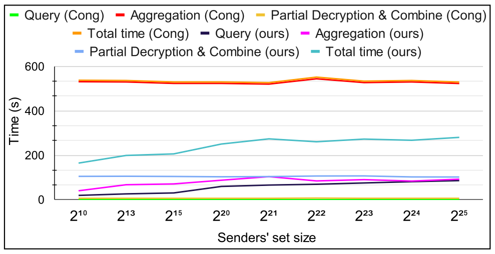


Figure 7: Computation time comparison for Cong *et al.* [38] vs ours for different senders' set size. Number of senders is set to 1024. **Bay *et al.* and Nevo *et al.* were omitted due to inefficiency for larger set sizes and a high number of senders, respectively.

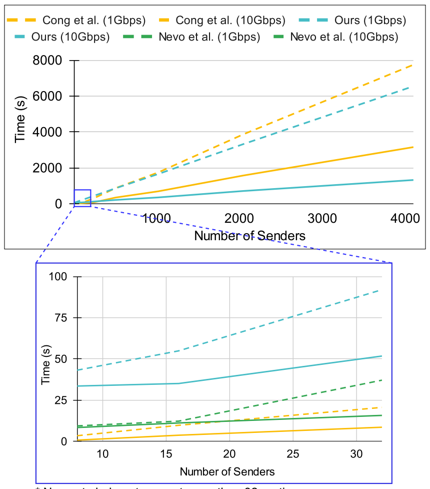
time is also taken into account for the overall latency. We observe this speedup due to our *summation* based approach for aggregation. Moreover, continuing to increase the number of senders results in an even greater computational latency benefit.

In terms of communication size, [38] has an advantage over our protocol since it uses smaller FHE parameters while query computations as they operate on unencrypted datasets (Table 4). We, however, process fully encrypted datasets and provide security against the senders that necessitate the use of non-scaler homomorphic operations requiring relatively larger FHE parameters. Moreover, our communication cost scales much better when dealing with a high number of senders, as depicted in Figure 8. Our communication is mainly dependent on $|\mathcal{X}|$ and grows slowly with the increasing number of senders. Considering $l \in [2^9, 2^{13}]$ and

Table 4: Communication cost for our protocol compared to Cong *et al.* [38] and Nevo *et al.* [84] for up to $l = 15$, $\alpha = l - 1$ and 2^{20} senders' set size. "Total Comm. size" refers to the communication size of sent/received data between all parties[1,2].
****Bay *et al.* [8] omitted as they lack an implementation level communication analysis and cannot handle 2^{20} senders' set size.**

Params.		Total Comm. size			Comm. time (seconds)								
					10 Gbps			1 Gbps			200 Mbps		
l	α	[38]	[84]	Ours	[38]	[84]	Ours	[38]	[84]	Ours	[38]	[84]	Ours
4	1	180.1	201.3	759.3	0.5	0.2	0.6	1.8	1.6	6.1	7.5	9.0	30.7
	3	195.5	2225.6	771.7	0.5	1.8	0.6	1.9	17.8	6.2	8.1	89.3	31.2
10	1	497.2	453.0	2043.3	0.7	0.4	1.6	4.3	3.6	16.3	20.2	18.4	82.1
	4	515.8	1054.6	2061.9	0.7	0.8	1.7	4.5	8.4	16.5	21.1	42.5	82.8
	9	546.8	5563.9	2082.9	0.8	4.5	1.7	4.7	44.5	16.7	22.2	222.9	84.0
15	1	742.7	662.7	3113.3	0.9	0.5	2.5	6.3	5.3	24.9	30.0	26.8	124.9
	4	761.3	1254.6	3131.9	0.9	1.0	2.5	6.4	10.0	25.1	30.8	50.5	125.6
	7	779.9	1416.9	3150.5	1.0	1.1	2.5	6.6	11.3	25.2	31.5	57.0	126.3
	14	823.3	8345.9	3193.6	1.0	6.7	2.6	6.9	66.8	25.6	33.3	334.2	128.1

1 [38] and [84] do not provide security of datasets against the senders.
2 [84] do not support sender side encryption. Implementing sender side encryption in [38] results in impractical runtimes (multiple hours or days) due to huge overheads for optimization operations and intersection polynomial generation for interpolation in the homomorphic domain.



* Nevo et al. do not support more than 32 parties

Figure 8: Total runtime comparison for Cong *et al.* [38] and Nevo *et al.* [84] with ours. Senders' set size is set to 2^{15} and $\alpha = \frac{l}{2}$.
****Bay *et al.* omitted as they do not support 2^{15} set size.**

high bandwidth network environments (around 1Gbps or more), our total runtime latency is up to 5.6× better than [38].

Comparison with non-FHE-based protocols. For non-FHE-based protocols, multiplicative depth is not a significant issue, but most of these protocols have high communication and no post-quantum security. These protocols either support a large senders' set size or a large number of senders but not both. We compared both our work and [38] to Bay *et al.* [8] and Nevo *et al.* [84] in the MPSI setting with $\alpha = \frac{l}{2}$ collusion threshold.

In terms of communication, [8] require five rounds while ours and [84] need four. [8] only provide a theoretical level communicational analysis, lacking an implementation level evaluation, thus, for communication we only compare ours with [38] and [84]. Theoretically, clients in [8] send encrypted bloom filters comprising m ciphertexts, where m is determined by the number of hash functions h and the dataset size n. In experiments, [8] set $h = 7$ and $m = \lfloor \frac{7n}{\log 2} \rfloor$ bits to achieve a 1% false positive rate, while our method maintains a negligible false positive rate. For sensitive applications requiring a much lower false positive rate, [8] must use higher m and h, which will incur significantly larger overheads.

[8, 38, 84] access the senders' set elements in plain for encoding while we operate on encrypted sets. In [8], senders can have duplicate elements among each other, but in ours, we assume that all senders have distinct elements. Senders' set size is set to 2^7 for [8], but for the number of parties higher than 128, 256, and 512, we set the senders' set size to 2^8, 2^9, and 2^{10} for us and [38], [84] respectively even though it favors [8]. We report the runtime comparison using these settings in Figure 6 (middle, left-zoomed).

[84] are extremely fast as they employ very efficient OPPRF and OKVS-based primitives; however, they can only handle up to 32 parties. For applications involving a limited number of parties and malicious security, [84] is preferable. [38] and [8] are better for applications having less than around 300 parties but [8] are limited in set size. Our protocol scales much better and is up to 1.5× and 2.4× faster than [8] and [38] for up to 2^{10} senders. Hence, ours is preferable for applications with a very high number of parties that require security against the senders.

Communication and total runtime. We compare our protocol to [84] and [38] in Table 4 for communication size. We limit $l = 15$ as [84] only provide communication analysis for up to 15 parties in [[84], Table 4]. Our protocol outperforms [84] when the collusion level reaches $\alpha = l - 1$, though [84] excels at lower α values. [38] surpasses both our method and [84] in terms of communication only. In Figure 8, we show the total runtime latency of our protocol compared to [38] and [84]. Even though [38] and [84] have lower latencies for a smaller number of parties (due to unencrypted senders' sets), ours is up to 2.4× faster for a higher number of parties as we rely on *summation*-based aggregation.

Comparison with other works. [71] is close to our work in the MPSI setting; however, it scales poorly for a large number of parties, taking almost 300 seconds of computation time for 15 parties. [9] provide a maliciously secure MPSI based on garbled bloom filters and k-out-of-N OT. [9] and [71] both has been compared against [84] hence we do not compare against [9] and [71]. [24] employ a multi-query reverse private membership test protocol, but their implementation is not compatible with multiple parties. [112] achieve maliciously secure MPSI using bloom filters for large inputs; however, they lack a public implementation for comparison.

7 CONCLUSION

In this work, we introduce the concept of a Private Segmented Membership Test (PSMT) for cases where users wish to query a set held segmented among many data holders while ensuring provenance privacy. We show a basic protocol to solve PSMT based on approximate-arithmetic threshold FHE and provide details about overcoming various technical challenges to make our solution feasible. We further guarantee IND-CPA^D security for threshold CKKS using noise-smudging techniques. Our experiment shows the scalability of our protocol that aggregates the penultimate results from data holders. In future work, we aim to explore new avenues for further optimizations, discussions, and better communication.

ACKNOWLEDGMENTS

This material is based upon work supported by the U.S. Department of Homeland Security under Grant Award Number 17STQAC0000 1-07-00. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Department of Homeland Security. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

We would like to thank Yuriy Polyakov, Kurt Rohloff, David Cousins of Duality Inc., and Nick Genise of Apple Inc. (previously at Duality Inc.) for their helpful comments and conversations. Additionally, we are grateful to Professor Jung Hee Cheon, Mr. Wootae Kim, and Mr. Jai Hyun Park from Seoul National University for their valuable insights related to Domain Extension Polynomials. Finally, we would like to thank the anonymous reviewers for their helpful comments and suggestions.

REFERENCES

[1] 2021. GitHub - microsoftAPSI: APSI is a C++ library for Asymmetric (unlabeled or labeled) Private Set Intersection. — github.com. https://github.com/microsoft/APSI

[2] Ahmad Al Badawi, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, et al. 2022. OpenFHE: Open-source fully homomorphic encryption library. In *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. 53–63.

[3] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, et al. 2021. Homomorphic encryption standard. *Protecting privacy through homomorphic encryption* (2021), 31–62.

[4] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. 2018. *Homomorphic Encryption Security Standard*. Technical Report. HomomorphicEncryption.org, Toronto, Canada.

[5] Andreea Alexandru, Ahmad Al Badawi, Daniele Micciancio, and Yuriy Polyakov. 2024. Application-Aware Approximate Homomorphic Encryption: Configuring FHE for Practical Use. *Cryptology ePrint Archive* (2024).

[6] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. 2012. Multiparty computation with low communication, computation and interaction via threshold FHE. In *Advances in Cryptology–EUROCRYPT 2012: 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings 31*. Springer, 483–501.

[7] Saikrishna Badrinarayanan, Peihan Miao, Srinivasan Raghuraman, and Peter Rindal. 2021. Multi-party threshold private set intersection with sublinear communication. Springer, 349–379.

[8] Aslı Bay, Zekeriya Erkin, Jaap-Henk Hoepman, Simona Samardjiska, and Jelle Vos. 2021. Practical multi-party private set intersection protocols. *IEEE Transactions on Information Forensics and Security* 17 (2021), 1–15.

[9] Aner Ben-Efraim, Olga Nissenbaum, Eran Omri, and Anat Paskin-Cherniavsky. 2022. Psimple: Practical multiparty maliciously-secure private set intersection. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*. 1098–1112.

[10] Song Bian, Zhou Zhang, Haowen Pan, Ran Mao, Zian Zhao, Yier Jin, and Zhenyu Guan. 2023. HE3DB: An Efficient and Elastic Encrypted Database Via Arithmetic-And-Logic Fully Homomorphic Encryption. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 2930–2944.

[11] Derian Boer, Zahra Ahmadi, and Stefan Kramer. 2019. Privacy preserving client/vertical-servers classification. In *ECML PKDD 2018 Workshops: MIDAS 2018 and PAP 2018, Dublin, Ireland, September 10-14, 2018, Proceedings 3*. Springer, 125–140.

[12] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter MR Rasmussen, and Amit Sahai. 2018. Threshold cryptosystems from threshold fully homomorphic encryption. In *Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part I 38*. Springer, 565–596.

[13] Joppe W Bos, Kristin Lauter, and Michael Naehrig. 2014. Private predictive analysis on encrypted medical data. *Journal of biomedical informatics* 50 (2014), 234–243.

[14] John P Boyd and Fei Xu. 2009. Divergence (Runge phenomenon) for least-squares polynomial approximation on an equispaced grid and Mock–Chebyshev subset interpolation. *Appl. Math. Comput.* 210, 1 (2009), 158–168.

[15] Zvika Brakerski. 2012. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Advances in Cryptology–CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*. Springer, 868–886.

[16] Zvika Brakerski, Craig Gentry, and Shai Halevi. 2013. Packed ciphertexts in LWE-based homomorphic encryption. In *Public-Key Cryptography–PKC 2013: 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26–March 1, 2013. Proceedings 16*. Springer, 1–13.

[17] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2014. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)* 6, 3 (2014), 1–36.

[18] Nishanth Chandran, Nishka Dasgupta, Divya Gupta, Sai Lakshmi Bhavana Obbattu, Sruthi Sekar, and Akash Shah. 2021. Efficient Linear Multiparty PSI and Extensions to Circuit/Quorum PSI. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 1182–1204.

[19] Marina Checri, Renaud Sirdey, Aymen Boudguiga, Jean-Paul Bultel, and Antoine Choffrut. 2024. On the practical CPAD security of "exact" and threshold FHE schemes and libraries. *Cryptology ePrint Archive* (2024).

[20] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H Lai. 2019. Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 142–157.

[21] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. 2018. Labeled PSI from fully homomorphic encryption with malicious security. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1223–1237.

[22] Hao Chen, Kim Laine, and Rachel Player. 2017. Simple encrypted arithmetic library-SEAL v2. 1. In *Financial Cryptography and Data Security: FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers 21*. Springer, 3–18.

[23] Hao Chen, Kim Laine, and Peter Rindal. 2017. Fast private set intersection from homomorphic encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1243–1255.

[24] Yu Chen, Min Zhang, Cong Zhang, Minglang Dong, and Weiran Liu. 2022. Private Set Operations from Multi-Query Reverse Private Membership Test. *Cryptology ePrint Archive* (2022).

[25] Jung Hee Cheon, Hyeongmin Choe, Alain Passelègue, Damien Stehlé, and Elias Suvanto. 2024. Attacks Against the INDCPA-D Security of Exact FHE Schemes. *Cryptology ePrint Archive* (2024).

[26] Jung Hee Cheon, Seungwan Hong, and Duhyeong Kim. 2020. Remark on the security of ckks scheme in practice. *Cryptology ePrint Archive* (2020).

[27] Jung Hee Cheon, Stanislaw Jarecki, and Jae Hong Seo. 2012. Multi-party privacy-preserving set intersection with quasi-linear complexity. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 95, 8 (2012), 1366–1378.

[28] Jung Hee Cheon, Jinhyuck Jeong, Joohee Lee, and Keewoo Lee. 2017. Privacy-preserving computations of predictive medical models with minimax approximation and non-adjacent form. In *Financial Cryptography and Data Security: FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers 21*. Springer, 53–74.

[29] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*. Springer, 409–437.

[30] Jung Hee Cheon, Dongwoo Kim, and Duhyeong Kim. 2020. Efficient homomorphic comparison methods with optimal complexity. In *Advances in Cryptology–ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part II 26*. Springer, 221–256.

[31] Jung Hee Cheon, Dongwoo Kim, Duhyeong Kim, Hun Hee Lee, and Keewoo Lee. 2019. Numerical method for comparison on homomorphically encrypted numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 415–445.

[32] Jung Hee Cheon, Wootae Kim, and Jai Hyun Park. 2022. Efficient Homomorphic Evaluation on Large Intervals. *IEEE Transactions on Information Forensics and Security* 17 (2022), 2553–2568.

[33] Eduardo Chielle, Homer Gamil, and Michail Maniatakos. 2021. Real-time private membership test using homomorphic encryption. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1282–1287.

[34] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2020. TFHE: fast fully homomorphic encryption over the torus. *Journal of Cryptology*

33, 1 (2020), 34–91.

[35] Benny Chor, Niv Gilboa, and Moni Naor. 1997. Private information retrieval by keywords. (1997).

[36] Michele Ciampi and Claudio Orlandi. 2018. Combining private set-intersection with secure two-party computation. In *Security and Cryptography for Networks: 11th International Conference, SCN 2018, Amalfi, Italy, September 5–7, 2018, Proceedings*. Springer, 464–482.

[37] Valentina Ciriani, Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. 2009. Fragmentation design for efficient query execution over sensitive distributed databases. In *2009 29th IEEE International Conference on Distributed Computing Systems*. IEEE, 32–39.

[38] Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. 2021. Labeled PSI from homomorphic encryption with reduced computation and communication. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 1135–1150.

[39] Anamaria Costache, Benjamin R Curtis, Erin Hales, Sean Murphy, Tabitha Ogilvie, and Rachel Player. 2023. On the precision loss in approximate homomorphic encryption. In *International Conference on Selected Areas in Cryptography*. Springer, 325–345.

[40] Jack LH Crawford, Craig Gentry, Shai Halevi, Daniel Platt, and Victor Shoup. 2018. Doing real work with FHE: the case of logistic regression. In *Proceedings of the 6th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. 1–12.

[41] StackExchange Cryptography. 2022. *How to choose the large noise when using noise flooding technique in FHE?* https://crypto.stackexchange.com/questions/101010/how-to-choose-the-large-noise-when-using-noise-flooding-technique-in-fhe

[42] Morten Dahl, Daniel Demmler, Sarah El Kazdadi, Arthur Meyre, Jean-Baptiste Orfila, Dragos Rotaru, Nigel P Smart, Samuel Tap, and Michael Walter. 2023. Noah's Ark: Efficient Threshold-FHE Using Noise Flooding. In *Proceedings of the 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. 35–46.

[43] Emiliano De Cristofaro and Gene Tsudik. 2010. Practical private set intersection protocols with linear complexity. In *International Conference on Financial Cryptography and Data Security*. Springer, 143–159.

[44] Sumit Kumar Debnath and Ratna Dutta. 2016. Provably secure fair mutual private set intersection cardinality utilizing bloom filter. In *International Conference on Information Security and Cryptology*. Springer, 505–525.

[45] OpenFHE Discourse. 2022. *Appropriate error parameters for the noise flooding*. https://openfhe.discourse.group/t/appropriate-error-parameters-for-the-noise-flooding/95

[46] Changyu Dong, Liqun Chen, and Zikai Wen. 2013. When private set intersection meets big data: an efficient and scalable protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 789–800.

[47] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive* (2012).

[48] Michael J Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. 2005. Keyword Search and Oblivious Pseudorandom Functions.. In *TCC*, Vol. 3378. Springer, 303–324.

[49] Michael J Freedman, Kobbi Nissim, and Benny Pinkas. 2004. Efficient private matching and set intersection. In *International conference on the theory and applications of cryptographic techniques*. Springer, 1–19.

[50] Craig Gentry, Shai Halevi, and Nigel P Smart. 2012. Homomorphic evaluation of the AES circuit. In *Annual Cryptology Conference*. Springer, 850–867.

[51] Craig Gentry and Zulfikar Ramzan. 2005. Single-database private information retrieval with constant communication rate. In *International Colloquium on Automata, Languages, and Programming*. Springer, 803–815.

[52] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International conference on machine learning*. PMLR, 201–210.

[53] Oded Goldreich and Rafail Ostrovsky. 1996. Software protection and simulation on oblivious RAMs. *Journal of the ACM (JACM)* 43, 3 (1996), 431–473.

[54] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. 2017. Cache attacks on Intel SGX. In *Proceedings of the 10th European Workshop on Systems Security*. 1–6.

[55] Qian Guo, Denis Nabokov, Elias Suvanto, and Thomas Johansson. 2024. Key Recovery Attacks on Approximate Homomorphic Encryption with Non-Worst-Case Noise Flooding Countermeasures. In *33rd USENIX Security Symposium (USENIX Security 24). Philadelphia, PA: USENIX Association*.

[56] Shai Halevi. 2017. Homomorphic encryption. In *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich*. Springer, 219–276.

[57] Thang Hoang, Jorge Guajardo, and Attila A Yavuz. 2020. MACAO: A maliciously-secure and client-efficient active ORAM framework. *Cryptology ePrint Archive* (2020).

[58] Thang Hoang, Attila A Yavuz, and Jorge Guajardo. 2020. A multi-server oram framework with constant client bandwidth blowup. *ACM Transactions on Privacy and Security (TOPS)* 23, 1 (2020), 1–35.

[59] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, Mariana Raykova, David Shanahan, and Moti Yung. 2020. On deploying secure computing: Private intersection-sum-with-cardinality. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 370–389.

[60] Mihaela Ion, Ben Kreuter, Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, David Shanahan, and Moti Yung. 2017. Private intersection-sum protocol with applications to attributing aggregate ad conversions. *Cryptology ePrint Archive* (2017).

[61] Irina Ivanova. 2022. Rich Americans hide "Billions" offshore thanks to Tax Loophole, Senate panel finds. https://www.cbsnews.com/news/tax-evasion-billions-offshore-fatca-tax-reporting-loophole-senate-finance-committee-robert-brockman/

[62] Yuting Jiang, Jianghong Wei, and Jing Pan. 2022. Publicly verifiable private set intersection from homomorphic encryption. In *International Symposium on Security and Privacy in Social Networks and Big Data*. Springer, 117–137.

[63] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. 2019. Mobile private contact discovery at scale. In *28th USENIX Security Symposium (USENIX Security 19)*. 1447–1464.

[64] Tanveer Khan, Alexandros Bakas, and Antonis Michalas. 2021. Blind faith: Privacy-preserving machine learning using function approximation. In *2021 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 1–7.

[65] Eunkyung Kim, Jinhyuck Jeong, Hyojin Yoon, Younghyun Kim, Jihoon Cho, and Jung Hee Cheon. 2020. How to securely collaborate on data: Decentralized threshold he and secure key update. *IEEE Access* 8 (2020), 191319–191329.

[66] Miran Kim, Yongsoo Song, Shuang Wang, Yuhou Xia, Xiaoqian Jiang, et al. 2018. Secure logistic regression based on homomorphic encryption: Design and evaluation. *JMIR medical informatics* 6, 2 (2018), e8805.

[67] Lea Kissner and Dawn Song. 2005. Privacy-preserving set operations. In *Annual International Cryptology Conference*. Springer, 241–257.

[68] Kamil Kluczniak and Giacomo Santato. 2023. On Circuit Private, Multikey and Threshold Approximate Homomorphic Encryption. *Cryptology ePrint Archive* (2023).

[69] Vladimir Kolesnikov and Ranjit Kumaresan. 2013. Improved OT extension for transferring short secrets. In *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*. Springer, 54–70.

[70] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. 2016. Efficient batched oblivious PRF with applications to private set intersection. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 818–829.

[71] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. 2017. Practical multi-party private set intersection from symmetric-key techniques. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1257–1272.

[72] Anunay Kulshrestha and Jonathan Mayer. 2021. Identifying harmful media in {End-to-End} encrypted communication: Efficient private membership computation. In *30th USENIX Security Symposium (USENIX Security 21)*. 893–910.

[73] Eyal Kushilevitz and Rafail Ostrovsky. 1997. Replication is not needed: Single database, computationally-private information retrieval. In *Proceedings 38th annual symposium on foundations of computer science*. IEEE, 364–373.

[74] Thijs Laarhoven, Michele Mosca, and Joop Van De Pol. 2013. Solving the shortest vector problem in lattices faster using quantum search. In *Post-Quantum Cryptography: 5th International Workshop, PQCrypto 2013, Limoges, France, June 4-7, 2013. Proceedings 5*. Springer, 83–101.

[75] Baiyu Li and Daniele Micciancio. 2021. On the security of homomorphic encryption on approximate numbers. In *Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part I 40*. Springer, 648–677.

[76] Baiyu Li, Daniele Micciancio, Mark Schultz, and Jessica Sorrell. 2022. Securing approximate homomorphic encryption using differential privacy. In *Annual International Cryptology Conference*. Springer, 560–589.

[77] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. 2015. Last-level cache side-channel attacks are practical. In *2015 IEEE symposium on security and privacy*. IEEE, 605–622.

[78] Catherine Meadows. 1986. 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, 134–134.

[79] Daniele Micciancio and Oded Regev. 2009. Lattice-based cryptography. In *Post-quantum cryptography*. Springer, 147–191.

[80] Atsuko Miyaji, Kazuhisa Nakasho, and Shohei Nishida. 2017. Privacy-preserving integration of medical data: a practical multiparty private set intersection. *Journal of medical systems* 41 (2017), 1–10.

[81] Atsuko Miyaji and Shohei Nishida. 2015. A scalable multiparty private set intersection. In *International conference on network and system security*. Springer, 376–385.

[82] Daniel Morales, Isaac Agudo, and Javier Lopez. 2023. Private set intersection: A systematic literature review. *Computer Science Review* 49 (2023), 100567.
[83] Muhammad Haris Mughees and Ling Ren. 2023. Vectorized batch private information retrieval. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 437–452.
[84] Ofri Nevo, Ni Trieu, and Avishay Yanai. 2021. Simple, fast malicious multiparty private set intersection. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 1151–1165.
[85] John Newman and Ritchie Amy. 2023. Financial privacy rule. https://www.ftc.gov/legal-library/browse/rules/financial-privacy-rule
[86] Department of Homeland Security. 2022. DHS Use Cases of Privacy Enhancing Technologies. https://pets4hse.org/PETS4HSEUseCases.pdf
[87] OpenFHE. 2022. *OpenFHE Noise-Flooding and Static Noise Estimation*. https://github.com/openfheorg/openfhe-development/blob/main/src/pke/examples/CKKS_NOISE_FLOODING.md
[88] Michele Orrù, Emmanuela Orsini, and Peter Scholl. 2017. Actively secure 1-out-of-N OT extension with application to private set intersection. In *Topics in Cryptology–CT-RSA 2017: The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14–17, 2017, Proceedings*. Springer, 381–396.
[89] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. 2015. Phasing: Private set intersection using permutation-based hashing. In *24th USENIX Security Symposium (USENIX Security 15)*. 515–530.
[90] Benny Pinkas, Thomas Schneider, Nigel P Smart, and Stephen C Williams. 2009. Secure two-party computation is practical. In *Advances in Cryptology–ASIACRYPT 2009: 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings 15*. Springer, 250–267.
[91] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. 2019. Efficient circuit-based PSI with linear communication. In *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part III 38*. Springer, 122–153.
[92] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. 2018. Efficient circuit-based PSI via cuckoo hashing. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 125–157.
[93] Benny Pinkas, Thomas Schneider, and Michael Zohner. 2014. Faster private set intersection based on {OT} extension. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*. 797–812.
[94] Benny Pinkas, Thomas Schneider, and Michael Zohner. 2018. Scalable private set intersection based on OT extension. *ACM Transactions on Privacy and Security (TOPS)* 21, 2 (2018), 1–35.
[95] Raluca Ada Popa, Catherine MS Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2011. CryptDB: Protecting confidentiality with encrypted query processing. In *Proceedings of the twenty-third ACM symposium on operating systems principles*. 85–100.
[96] The Washington Post. 2023. Foreign banks to help U.S. fight tax evasion. https://www.washingtonpost.com/business/economy/foreign-banks-to-help-us-fight-tax-evasion/2014/06/02/52b3919c-ea92-11e3-93d2-edd4be1f5d9e_story.html
[97] Sara Ramezanian, Tommi Meskanen, Masoud Naderpour, Ville Junnila, and Valtteri Niemi. 2020. Private membership test protocol with low communication complexity. *Digital Communications and Networks* 6, 3 (2020), 321–332.
[98] Peter Rindal and Mike Rosulek. 2016. Faster malicious 2-party secure computation with {Online/Offline} dual execution. In *25th USENIX Security Symposium (USENIX Security 16)*. 297–314.
[99] Peter Rindal and Mike Rosulek. 2017. Improved private set intersection against malicious adversaries. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 235–259.
[100] Jason Schlessman. 2002. *Approximation of the sigmoid function and its derivative using a minimax approach*. Ph. D. Dissertation. Lehigh University.
[101] Nigel P Smart and Frederik Vercauteren. 2014. Fully homomorphic SIMD operations. *Designs, codes and cryptography* 71 (2014), 57–81.
[102] Yongha Son and Jinhyuck Jeong. 2023. PSI with computation or Circuit-PSI for Unbalanced Sets from Homomorphic Encryption. *Cryptology ePrint Archive* (2023).
[103] Sandeep Tamrakar, Jian Liu, Andrew Paverd, Jan-Erik Ekberg, Benny Pinkas, and N Asokan. 2017. The circle game: Scalable private membership test using trusted hardware. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. 31–44.
[104] Adithya Vadapalli, Ryan Henry, and Ian Goldberg. 2023. Duoram: A Bandwidth-Efficient Distributed ORAM for 2-and 3-Party Computation. In *32nd USENIX Security Symposium*.
[105] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient {Out-of-Order} execution. In *27th USENIX Security Symposium (USENIX Security 18)*. 991–1008.
[106] Ke Coby Wang and Michael K Reiter. 2018. How to end password reuse on the web. *arXiv preprint arXiv:1805.00566* (2018).
[107] Wenli Wang, Shundong Li, Jiawei Dou, and Runmeng Du. 2020. Privacy-preserving mixed set operations. *Information Sciences* 525 (2020), 67–81.
[108] Zhusheng Wang, Karim Banawan, and Sennur Ulukus. 2021. Multi-party private set intersection: An information-theoretic approach. *IEEE Journal on Selected Areas in Information Theory* 2, 1 (2021), 366–379.
[109] Lifei Wei, Jihai Liu, Lei Zhang, and Wuji Zhang. 2022. Efficient and collusion resistant multi-party private set intersection protocols for large participants and small sets setting. In *International Symposium on Cyberspace Safety and Security*. Springer, 118–132.
[110] ACC Yao. 1986. How to generate and exchange secrets. In—27th Annual Symposium on Foundations of Computer Science.
[111] Jason HM Ying, Shuwei Cao, Geong Sen Poh, Jia Xu, and Hoon Wei Lim. 2022. PSI-stats: private set intersection protocols supporting secure statistical functions. In *International Conference on Applied Cryptography and Network Security*. Springer, 585–604.
[112] En Zhang, Feng-Hao Liu, Qiqi Lai, Ganggang Jin, and Yu Li. 2019. Efficient multi-party private set intersection against malicious adversaries. In *Proceedings of the 2019 ACM SIGSAC conference on cloud computing security workshop*. 93–104.

A PROTOCOL PARAMETERS

We analyzed various levels of parameters for the DEP procedure and Chebyshev-based approximation and how they affect the performance and accuracy of our protocol for PSMT. Using $B(x) = x - \frac{4}{27}x^3$, it is imperative to maintain L below $1.5 \times \sqrt{3}$ while using Domain Extension Polynimials [32]. Thus, for expanding the domain range, it requires increasing R and n so that $[-L^n R, L^n R]$ adequately accommodates large input sets without amplifying errors and computation. However, using arbitrary large values for R and n would require very high depths for homomorphic computation and amplify errors in the approximation. Using smaller values of L, especially close to 1.5, greatly increases the accuracy of the DEP method, but this results in larger values for other parameters, incurring very high computational costs.

The optimal DEP, Chebyshev, and optimization parameters for running our protocol for various senders' set sizes are provided in Table 5. K and S are set to 1 and 10, respectively, for the VAF. $|\mathcal{X}|$ denotes the size of the senders's set. "Non-intersection" denotes the summation value of ciphertext's slots when there is no intersection, and "Intersection" similarly denotes the summation value of ciphertext's slots during an intersection. To confirm an intersection for $l < 4096$, setting threshold $\tau > 200$ is sufficient for the parameters described in Table 5. A higher value of τ handles the precision loss occurring due to the additional added noise from noise-smudging in cases where $z = \sum_{i=1}^{l-1} etan_i$ overflow beyond negligible values. The value τ can be adjusted to higher values for a higher number of parties. We note that the parameters we provide are highly optimized for the particular computations, and some values in Table 5 were found empirically.

Table 5: Our protocol parameters to solve PSMT for various senders' set sizes

$\|\mathcal{X}\|$	DEP			Chebyshev Parameters		Optimization			Aggregation Result	
	L	R	n	c	κ	j	k	ρ	Non-intersection	Intersection
2^7	2.50	21.0	2	27	3.4×10^{-5}	3	3	2.5	3.1×10^{-5}	1528.18
2^8	2.56	16.0	3	27	3.2×10^{-1}	1	3	2.5	2.9×10^{-1}	1526.54
2^{10}	2.58	24.0	4	27	1.8×10^{-6}	4	3	2.5	1.1×10^{-4}	1525.88
2^{13}	2.58	27.5	6	27	5.6×10^{-3}	4	3	2.5	4.7×10^{-3}	1525.88
2^{15}	2.58	43.5	7	27	3.7×10^{-1}	4	3	2.5	4.9×10^{-1}	1526.63
2^{20}	2.59	200.0	9	247	1.2×10^{-1}	2	3	2.5	2.2×10^{-1}	1526.11
2^{21}	2.59	400.0	9	247	7.3×10^{-1}	4	3	2.7	8.1×10^{-1}	2824.66
2^{22}	2.59	800.0	9	247	7.6×10^{-1}	6	3	2.7	8.8×10^{-1}	2824.29
2^{23}	2.59	1600.0	9	247	6.3×10^{-1}	8	3	2.7	9.1×10^{-1}	2823.91
2^{24}	2.59	3200.0	9	247	7.9×10^{-1}	10	3	2.7	8.8×10^{-1}	2824.20
2^{25}	2.59	6400.0	9	247	2.7×10^{-1}	12	3	2.7	7.4×10^{-1}	2823.92