

Mercury: Fuzzy Private Set Intersection with Communication and Computation Logarithmic in the Threshold

Qi Liu¹, Kang Yang², Binwu Xiang³, Xiaojie Guo⁴, Cheng Hong⁵, and Yu Yu¹

¹ Shanghai Jiao Tong University, Shanghai, China
sjtu-liuqi@sjtu.edu.cn, yyuu@sjtu.edu.cn

² State Key Laboratory of Cryptology, Beijing, China
yangk@sklc.org

³ East China Normal University, Shanghai, China
bwxiang@sc.ecnu.edu.cn

⁴ Shanghai Qi Zhi Institute, Shanghai, China
guoxiaojie@sqz.ac.cn

⁵ Ant Group, Beijing, China
vince.hc@antgroup.com

Abstract. Private Set Intersection (PSI) allows two mutually distrusting parties to compute the intersection of their sets without revealing any additional information and has found numerous applications. The approximate matching required by various applications has led to the development of Fuzzy PSI (FPSI), a variant of PSI that identifies approximate intersection elements under a public distance threshold δ . Reducing computation and communication costs remains an important challenge for FPSI. In this paper, we present a new FPSI protocol, called Mercury, which achieves the logarithmic computation and communication complexity with respect to threshold δ . Our protocol exploits the correlation of the intersection and the point pairs held by the parties, and efficiently identifies the neighboring point pairs by a mixed protocol from homomorphic encryption and secure two-party computation using a “mask-then-compare” technique. We implemented Mercury and evaluated both its sender-private and standard FPSI variants. For balanced two-dimensional inputs, Mercury-SP achieves $9.0 \sim 63.9\times$ runtime speedups and reduces communication by a factor of $8.7 \sim 26.1$ across the evaluated thresholds and set sizes in the LAN setting, compared to the FPSI protocol by Dang et al. (CCS’25). In the completed unbalanced experiments, Mercury-SP achieves $55.7 \sim 132.4\times$ WAN runtime speedups compared to the FPSI protocol by Piske et al. (CCS’25).

Keywords: Fuzzy private set intersection · homomorphic encryption · secure two-party computation

1 Introduction

Private set intersection (PSI) allows two parties to compute the intersection of their private sets without revealing any additional information. Initially in-

Table 1. Asymptotic complexities of L_2 FPSI protocols. Here, N_X and N_Y denote the input sizes, d the dimension, and δ the distance threshold. Parenthesized tags summarize the input conditions. The Y -sep. tag constrains only the receiver input, whereas disj. proj., 1-sep., and apart constrain both inputs. PST+25 assumes sender cell density one and public receiver neighborhood density ρ_η^Y , while cell-sparse denotes cell density one for both inputs. Exact definitions and candidate bounds appear in Appendix D.

Protocol (condition)	Communication	Computation	
		Sender	Receiver
vBaPu24 [2] (Y -sep.)	$O(N_X \delta^2 + d 2^d N_Y \delta)$	$O(N_X (d + \delta^2))$	$O(N_X + d 2^d N_Y \delta)$
GQL+24 [20] (disj. proj.)	$O(d\delta(N_X + N_Y))$	$O(dN_X \delta + N_Y)$	$O(N_X \log \delta + dN_Y \delta)$
DZL25 [14] (Y -sep.)	$O(2^d(N_X \log^d \delta + dN_Y \log \delta))$	$O(2^d N_X \log^d \delta)$	$O(2^d(N_X \log^d \delta + dN_Y \log \delta))$
ZCC+25 [62] (1-sep.)	$O(d\delta(N_X + N_Y))$	$O(d\delta N_X + N_Y)$	$O(N_X \log \delta + d\delta N_Y)$
vBaPu25 [56] (apart)	$O(d\delta N_X + (d + \log \delta) 2^d N_Y)$	$O(d\delta N_X + (d + \log \delta) 2^d N_Y)$	$O((d + \log \delta) 2^d N_Y)$
YHW+26 [59] (disj. proj.)	$O(d \log \delta (N_X + N_Y))$	$O(d \log \delta (N_X + N_Y))$	$O(d \log \delta (N_X + N_Y))$
PST+25 [43]	$O(d(N_X \delta + 2^d N_Y) \rho_\eta^Y)$	$O(dN_X \delta \rho_\eta^Y)$	$O((d 2^d N_Y + N_X) \rho_\eta^Y)$
Mercury-SP (cell-sparse)	$O(2^d C_{\text{pt}} N_X + N_Y \log \delta)$	$O((2^d N_X + C_{\text{pt}} N_Y) \log \delta)$	$O((2^d N_X + C_{\text{pt}} N_Y) \log \delta)$

roduced by Meadows [35], numerous efficient and practical PSI protocols have been subsequently proposed, e.g. [40, 29, 38, 10, 39, 49, 47, 45, 22, 5]. Recently, PSI is widely utilized in various real-world applications, including password breach detection [13], private contact discovery [28], secure genome testing [3], and on-line advertising measurement [36].

Fuzzy PSI (FPSI), first mentioned by [18], is a PSI variant proposed to identify the intersection element with approximate matching, rather than requiring exact equality. Recently, this variant has seen significant advancements for various distance metrics [11, 55, 23, 24, 2, 20, 21]. Beyond classic applications such as biometric identification [55] and image matching techniques [32], FPSI for L_2 distance additionally facilitates privacy-preserving face recognition [50], analysis of geometric coordinates or 3D features [23], and Deep Metric Learning [57]. As a concrete example, consider location-based private queries: the receiver holds a set of his current locations, and the sender has a set of restaurant locations. Using FPSI, the receiver can discover nearby restaurants (within L_2 distance) without learning all restaurant locations, while the sender knows nothing about the receiver’s locations.

Several L_2 FPSI constructions incur enumeration costs that grow at least linearly with the distance threshold δ [2, 20, 43]. These costs can become substantial as δ increases. Structure-aware PSI constructions [23, 24, 21] focus on L_1 and L_∞ distances. We study how to combine spatial hashing with secure distance testing to obtain concretely efficient L_2 FPSI with logarithmic dependence on δ under two-sided cell sparsity. Here and throughout, $N_X = |X|$ and $N_Y = |Y|$ denote the sender and receiver input sizes.

1.1 Our Contribution

In this paper, we propose Mercury, an efficient FPSI framework leveraging Homomorphic Encryption (HE) and Secure Two-Party Computation (2PC) for the

L_2 distance metric. We support both sender-private FPSI (FPSI-SP), where the receiver learns which of its points have a sender neighbor, and standard FPSI (Non-SP), where it learns the sender points having a receiver neighbor.

For both variants, the main challenge is to identify neighboring point pairs efficiently without revealing information beyond the prescribed output. To address this challenge, **Mercury-SP** uses spatial hashing to reduce the candidate set and mask-then-compare to perform secure threshold tests whose cost per candidate is logarithmic in δ .

Theoretical comparison. Table 1 compares the asymptotic complexities of L_2 FPSI protocols under their respective input conditions. For the balanced and unbalanced settings $N_X \geq N_Y$, we state the **Mercury-SP** bounds at fixed low dimension and security parameters, counting cryptographic, coordinate, and address words as unit-size encodings. Under two-sided cell sparsity, **Mercury-SP** has communication complexity $O(2^d C_{\text{pt}} N_X + N_Y \log \delta)$ and computation complexity $O((2^d N_X + C_{\text{pt}} N_Y) \log \delta)$ for each party in this model. Here, C_{pt} is the public capacity to which each sender point-table bin is padded, chosen to make the probability of sender-bin overflow negligible. The input conditions used by the prior protocols and their effects on candidate generation are discussed in Appendix D. The standard variant uses the same number of distance evaluations and range tests, with address and output costs accounted for separately in Appendix E.2.

For fixed d and C_{pt} , the bounds for **Mercury-SP** are linear in the input sizes and have at most logarithmic dependence on δ . Under the respective input conditions in Table 1, this threshold dependence improves on the linear or quadratic factors in [2, 20, 43] and the $O(\log^d \delta)$ factor in [14]. Yang et al. [59] also achieve logarithmic dependence on δ , but under a two-sided disjoint-projection condition rather than the two-sided cell sparsity used by **Mercury**. In terms of supported dimensions, the L_2 construction of Piske et al. [43] is restricted to $d = 2$, whereas **Mercury** supports $d = O(\log \lambda)$ for security parameter λ . We focus on low-dimensional geometric matching, such as proximity queries on two- or three-dimensional coordinates.

Implementation and performance evaluation. We implement both variants of **Mercury** in C++ and evaluate their L_2 performance across input sizes, dimensions, and thresholds in LAN and WAN settings. We compare with the open-source FPSI implementations of [2, 20, 43, 14, 59]. For balanced two-dimensional inputs (i.e., $N_X = N_Y$), **Mercury-SP** achieves $9.0 \sim 63.9\times$ runtime speedups and reduces communication by a factor of $8.7 \sim 26.1$ compared to [14] across the evaluated thresholds $32 \sim 128$ and set sizes $2^8 \sim 2^{12}$ in the LAN setting. Across these balanced two-dimensional configurations, **Mercury-SP** reduces communication by $1.9 \sim 5.6\times$ compared with the basic non-prefix implementation of Yang et al. [59], whose reported online times exclude input-dependent preprocessing. For unbalanced two-dimensional inputs with $N_Y = 2^{10}$ and $N_X \in \{2^{16}, 2^{18}\}$, **Mercury-SP** achieves $77.6 \sim 162.3\times$ LAN runtime speedups and $55.7 \sim 132.4\times$ WAN runtime speedups compared to [43] at receiver neighborhood density $\rho_\eta^Y = 9$. Section 5 details the benchmark configurations and timing scopes.

1.2 Related Work

FPSI was initiated by Freedman et al. [18] for approximate matching under Hamming distance. Subsequent works improved the security and efficiency of this line [11, 60, 55, 9, 12, 20]. Chakraborti et al. [9] introduced distance-aware PSI, targeting Hamming distance and one-dimensional L_1 distance over integers. Their protocol uses prefix representations to encode the range $[a - \delta, a + \delta]$ with logarithmic dependence on the threshold, a technique later used in [20]. These techniques are highly effective for Hamming distance and one-dimensional ranges, but extending them to multidimensional L_2 distance introduces a dimension-dependent blow-up.

Separately Garimella et al. [23, 24] introduced structure-aware PSI (sa-PSI) for L_1 and L_∞ distances using Boolean function secret sharing and oblivious transfer (OT), and later strengthened the construction against malicious adversaries. In their framework, Boolean Function Secret Sharing (FSS) describes the sender’s geometric structure, while OT enables the receiver to privately evaluate membership in that structure. Subsequently, Garimella et al. [21] proposed incremental bFSS with a more succinct representation, and Bui et al. [7] removed hidden multiplicative overhead in sa-PSI. This line gives efficient protocols for structured matching, especially for L_∞ , but it does not directly provide an efficient L_2 instantiation.

For geometric FPSI, van Baarsen and Pu [2] constructed protocols for general L_∞ and L_p distances by encoding potential intersection points and aggregating fuzzy-matching results with homomorphic encryption. Their construction encodes potential intersection points into DDH tuples and uses additive homomorphic encryption to combine matching results. Their later OPRF/OPPRF-based construction [56] improves concrete efficiency for L_1 and L_∞ by replacing heavier homomorphic-encryption components. Gao et al. [20] proposed fuzzy mapping, which uses spatial partitioning to align candidate points under a disjoint-projection assumption. Richardson et al. [46] route proximity-test subprotocols through a disjoint-hashing structure while hiding the bin containing the input point; this assumption was later adapted in [43]. These constructions use different input conditions to control candidate generation, as discussed in Appendix D.

Recent protocols seek to improve the efficiency of FPSI for L_2 distance. Piske et al. [43] instantiate L_2 FPSI through distance-OT, but their construction is specialized to $d = 2$ and has linear dependence on δ . Dang et al. [14] reduce the threshold dependence using prefix representations, but the cost scales as $O(\log^d \delta)$. Yang et al. [59] obtain logarithmic threshold dependence using shared-output OPRFs under a two-sided disjoint-projection condition. Under two-sided cell sparsity, **Mercury** combines spatial hashing, circuit-private HE distance computation, and mask-then-compare to achieve logarithmic threshold dependence in low-dimensional L_2 matching.

Finally, Son et al. [53] study high-dimensional similarity search with CKKS-based fully homomorphic encryption and support cosine-similarity-style matching. They compute the similarity score and compare it with the threshold directly over ciphertexts. Their setting is complementary to ours: it targets high-

dimensional vectors, but requires $O(N_X N_Y)$ pairwise computation. We focus instead on low-dimensional geometric FPSI, where the goal is to avoid all-pairs distance evaluation and keep the cost near-linear in the input size.

2 Technical Overview

2.1 Previous Approach

Recent FPSI protocols for the L_2 distance have made notable progress, but several limitations remain. First, the L_2 instantiation presented by Piske et al. [43] is specified for $d = 2$. Their protocol introduces the distance-OT framework and simulates distance computation using shared-OT [42], which leads to $O(\delta)$ computation and communication. Second, Dang et al. [14] reduce the enumeration cost by adopting the prefix-representation strategy of Chakraborti et al. [9], improving the dependence on δ to $O(\log^d \delta)$. A shared-output OPRF approach [59] also achieves logarithmic dependence on δ , under the two-sided disjoint-projection condition used in [20].

2.2 Our New Approach

Our starting point is to view fuzzy PSI through point pairs. The sender’s set X and the receiver’s set Y naturally induce

$$\text{Pairs} := \{(x, y) \mid x \in X, y \in Y\},$$

where $|\text{Pairs}| = N_X N_Y$. In classical PSI, each output element $z \in X \cap Y$ is witnessed by an exactly matching pair $(z, z) \in \text{Pairs}$. In fuzzy PSI, the witness is instead a *neighboring point pair* $(x, y) \in \text{Pairs}$ satisfying $\text{dist}(x, y) \leq \delta$. We focus on the sender-private construction **Mercury-SP**, in which the receiver learns only whether each of its points has a neighboring sender point. Appendix E presents the standard construction variant, which modifies the sender point table and output procedure to return sender points. We next describe the three modules of **Mercury-SP**: spatial hashing for candidate pair generation, batched encrypted distance evaluation, and mask-then-compare.

Spatial Hashing for Candidate Pair Generation The first stage uses the geometry of the input space to reduce the number of point pairs that require secure distance testing. To this end, we adopt the active-cell and spatial-hashing techniques of prior work [23, 2, 43]. We partition the input universe into input blocks and hash the active blocks into active-block tables BT_S and BT_R , using simple hashing for the sender and Cuckoo hashing for the receiver. When hashing succeeds, every neighboring point pair is represented by matching blocks in sender and receiver point-table bins with the same index. Consequently, **Mercury-SP** examines same-bin pairs rather than all pairs in Pairs . At fixed dimension, the number of real same-bin candidate pairs is $O(N_X)$. Encrypted

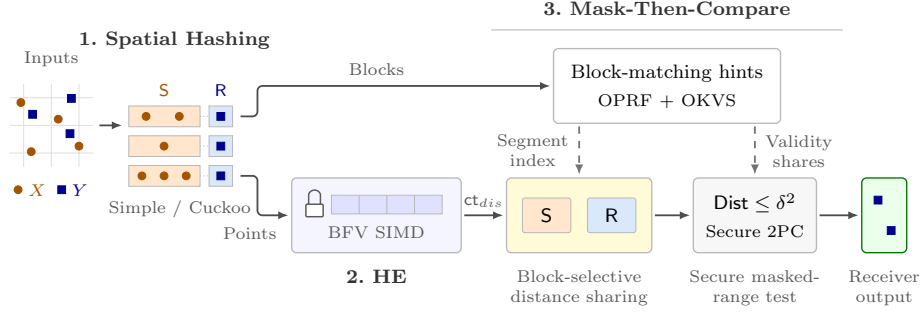


Fig. 1. Technical overview of Mercury-SP. Spatial hashing supplies the blocks and point pairs used for block matching and encrypted distance evaluation. Mask-then-compare combines the resulting hints and masked ciphertexts to return receiver points having a sender neighbor.

evaluation also processes padding, whose cost depends on the public sender-bin capacity C_{pt} selected in Section 5.1. The aligned candidates are then passed to batched encrypted distance evaluation and the mask-then-compare stage, which remove false positives without revealing the underlying distances.

Batched Encrypted Distance Evaluation After spatial hashing for candidate pair generation, the batched encrypted distance-evaluation module processes only same-bin candidate pairs. We use circuit-private linear HE [8] so that the sender can evaluate the distance circuit without learning the receiver’s input and without leaking its own private information. For a candidate pair (x, y) , the squared L_2 distance is computed as

$$\text{Dist}(x, y) = \sum_{\ell=1}^d (x[\ell] - y[\ell])^2 = \sum_{\ell=1}^d x[\ell]^2 + \sum_{\ell=1}^d y[\ell]^2 - 2 \sum_{\ell=1}^d x[\ell]y[\ell] \quad (1)$$

$$\text{dist}(x, y) = \sqrt{\text{Dist}(x, y)} \quad (2)$$

The protocol applies this identity to stored point offsets, whose distance equals the original point distance when the blocks match and hence have the same origin. The sender evaluates the cross terms using its packed point-table encoding pt_S and the receiver ciphertexts ct_R , and incorporates its squared-norm term and the masks described below into the same randomized evaluation. The receiver adds its own squared-norm term only after decrypting the masked response. Because the evaluated circuit depends on the sender’s private input, circuit privacy is essential for preventing the ciphertext from leaking sender-side information.

Mask-Then-Compare The same-bin candidates include false positives and padding, and revealing candidate distances would disclose information beyond the FPSI output. The mask-then-compare stage therefore supports threshold

testing without reconstructing candidate distances, while limiting secure comparisons to aligned candidate positions. The sender includes a distance pad and a distance-protection share in the evaluated expression for each real sender point-table entry. Within a bin, the receiver can cancel the distance pad only at aligned real positions of a sender block with the same origin and hash-function index.

This stage combines block-matching hints, block-selective distance sharing, and a secure masked-range test. For each bin i , block-matching hints provide the receiver with a segment-start index $\text{ST}_R[i]$ and give both parties secret shares of the match indicator and position-validity bits. When a matching sender block exists, this index identifies its point segment in $\text{PT}_S[i]$. Subsequent secure comparisons are restricted to the C_{act} candidate positions beginning at this index, with the validity bits excluding nonmatching and dummy positions from the output.

Using this index, block-selective distance sharing converts the selected masked evaluation results into additive shares for threshold testing. The sender incorporates the masks into the randomized evaluation, so the returned ciphertexts already encrypt masked values. For a double-active block, the parties obtain the same OPRF-derived key when their blocks have the same hash-function index. Domain-separated PRF evaluations under this key derive the distance pads, binding each pad to the bin and relative candidate position. For each bin, the sender samples an independent distance-protection share for each relative candidate position and reuses this vector across the sender blocks in that bin. The distance-protection share remains after the distance pad is cancelled, keeping the underlying distance hidden.

Finally, the parties apply the secure masked-range test to these shares and combine its secret-shared results with the validity bits in a 2PC circuit. For Mercury-SP, the receiver learns only the per-bin disjunction of the candidate predicates, not the individual candidate results or distances. Section 4.4 gives the composition of Mercury-SP, while Appendix E separately treats the candidate positions and sender points revealed by standard FPSI.

3 Preliminaries

3.1 Notation

Let λ and κ denote the computational and statistical security parameters, respectively, and $\text{negl}(\cdot)$ denote an unspecified negligible function. We denote $[a, b] := \{a, \dots, b\}$ and $[b] := [1, b]$. Let S and R denote the sender and the receiver, and let X and Y be the private sets of the sender and the receiver, respectively. We use a public finite coordinate domain $\mathcal{U} \subseteq \mathbb{Z}$, with $X, Y \subseteq \mathcal{U}^d$, and write $z[\ell]$ for the ℓ -th coordinate of a point z . $|X|$ represents the size of set X . We use $N_X = |X|$ and $N_Y = |Y|$ to denote the sizes of the sender's and receiver's inputs, respectively. We represent the dimension of the input space as d and the common number of bins in the aligned hash tables as m . We use $\langle \cdot \rangle_{\text{S,R}}$ to denote boolean shares where $\langle \cdot \rangle_{\text{S,R}} \in \{0, 1\}$ and $\llbracket \cdot \rrbracket_{\text{S,R}}$ to denote additive

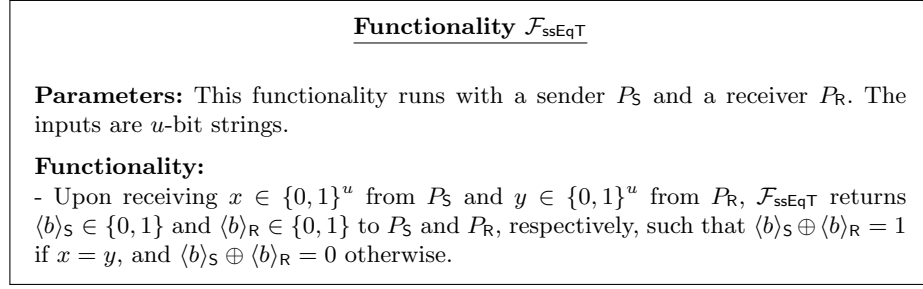


Fig. 2. The ideal functionality of secret-shared equality test

shares on $\mathbb{F}_p$ where $\llbracket \cdot \rrbracket_{S,R} \in \mathbb{F}_p$. Let C_{pt} denote the public padded capacity of a sender point-table bin, and let C_{act} denote the number of candidate positions processed per receiver bin. We write $\text{PT}_{S,R}$ for the pair of aligned point tables and $\text{BT}_{S,R}$ for the pair of aligned active-block tables. The receiver uses Cuckoo hashing to select one candidate bin for each active block, whereas the sender uses simple hashing to place each active block in all of its candidate bins. Thus, bin i in each pair refers to the same public hash-table index, even though the parties use different insertion rules.

3.2 Secure Two-Party Computation

A secure two-party computation (2PC) protocol allows two parties P_0 and P_1 , holding private inputs x_0 and x_1 , to compute a function output $f(x_0, x_1)$ without revealing additional information about their inputs. We consider static semi-honest adversaries and require that each party's view be simulatable from its own input and prescribed output [33].

We use two secret-shared 2PC subroutines. For a Boolean output b , each functionality samples $r \leftarrow_{\$} \{0,1\}$ and returns $\langle b \rangle_S = r$ and $\langle b \rangle_R = r \oplus b$ to the respective parties. Neither party learns b from its share alone. Unless stated otherwise, subsequent operations on Boolean shares, such as conjunction and disjunction, are evaluated within a secure 2PC circuit. The secret-shared equality test takes x from P_S and y from P_R , and returns boolean shares $\langle b \rangle_{S,R}$ of the equality bit. Its ideal functionality is shown in Fig. 2.

The secret-shared comparison protocol, also known as the Millionaires' protocol, takes x from P_S and y from P_R , and returns boolean shares of the comparison bit. The comparison functionality interprets its inputs as unsigned u -bit integers. Its ideal functionality is shown in Fig. 3.

We instantiate $\mathcal{F}_{\text{ssEqT}}$ and $\mathcal{F}_{\text{ssCmp}}$ using Cheetah [26] and use them to build the 2PC comparison component of Mercury.

Functionality $\mathcal{F}_{\text{ssComp}}$

Parameters: This functionality runs with a sender P_S and a receiver P_R . The inputs are unsigned u -bit integers represented in $\{0, 1\}^u$.

Functionality:

- Upon receiving $x \in \{0, 1\}^u$ from P_S and $y \in \{0, 1\}^u$ from P_R , $\mathcal{F}_{\text{ssComp}}$ returns $\langle b \rangle_S \in \{0, 1\}$ and $\langle b \rangle_R \in \{0, 1\}$ to P_S and P_R , respectively, such that $\langle b \rangle_S \oplus \langle b \rangle_R = 1$ if $x < y$, and $\langle b \rangle_S \oplus \langle b \rangle_R = 0$ otherwise.

Fig. 3. The ideal functionality of secret-shared comparison

3.3 Homomorphic Encryption

Homomorphic encryption (HE) [48, 25] enables computation on encrypted data without decryption. We use the BFV scheme [6, 16], which supports the linear homomorphic operations required by our distance computation.

BFV. For a power-of-two ring dimension N , plaintext modulus p , and ciphertext modulus q , define $R_p := \mathbb{Z}_p[X]/(X^N + 1)$ and $R_q := \mathbb{Z}_q[X]/(X^N + 1)$. We recall the BFV algorithms used in this work:

- **BFV.GenParam(1^λ)**: Given the computational security parameter λ , output encryption parameters $\text{param} = (N, p, q, \Delta, \mathcal{X}_{\text{sk}}, \mathcal{X}_e)$, where Δ is the scaling constant and $\mathcal{X}_{\text{sk}}, \mathcal{X}_e$ are the key and noise distributions.
- **BFV.KeyGen(param)**: Output a secret-public key pair (sk, pk) , where $\text{sk} \leftarrow \mathcal{X}_{\text{sk}}$, $\text{pk} \leftarrow (\text{pk}_0, \text{pk}_1) = (\text{pk}_0, \text{pk}_0 \cdot \text{sk} + e)$, $\text{pk}_0 \leftarrow_{\$} R_q$, and $e \leftarrow \mathcal{X}_e$.
- **BFV.Encrypt(param, sk, pt)**: Given a message $\text{pt} \in R_p$ and a secret key sk , output a ciphertext $\text{ct} = (c_0, c_1) = (c_0, c_0 \cdot \text{sk} + \Delta \cdot \text{pt} + e)$, where $c_0 \leftarrow_{\$} R_q$ and $e \leftarrow \mathcal{X}_e$.
- **BFV.Encrypt_p(param, pk, pt, e)**: Given a message $\text{pt} \in R_p$, a public key pk , and encryption noise $e = (e_0, e_1, e_2) \in R_q^3$, output $\text{ct} = e_2 \cdot \text{pk} + (e_0, \Delta \cdot \text{pt} + e_1)$.
- **BFV.Decrypt(param, sk, ct)**: Given a ciphertext $\text{ct} = (c_0, c_1)$, output $\text{pt} = \lceil (c_1 - c_0 \cdot \text{sk}) / \Delta \rceil \in R_p$.

As a public-key encryption scheme, BFV satisfies correctness and standard IND-CPA security.

The linear operations used in our construction are:

- (Addition) For two BFV ciphertexts ct_1, ct_2 and $\text{ct} \leftarrow \text{ct}_1 + \text{ct}_2$, it holds that $\text{BFV.Decrypt}(\text{ct}) = \text{BFV.Decrypt}(\text{ct}_1) + \text{BFV.Decrypt}(\text{ct}_2)$.
- (Plaintext Multiplication) For a BFV ciphertext ct_1 , a plaintext pt , and $\text{ct} \leftarrow \text{pt} \times \text{ct}_1$, it holds that $\text{BFV.Decrypt}(\text{ct}) = \text{pt} \times \text{BFV.Decrypt}(\text{ct}_1)$.

Linear Circuit Privacy. Standard IND-CPA security hides the encrypted message from the evaluator, but it does not ensure that an evaluated ciphertext hides the function applied to that message. In our distance computation, the coefficients of the evaluated linear function depend on the sender's private input. We therefore additionally require linear circuit privacy.

Definition 1 (Linear Circuit Privacy [8, Definition 3]). *A linearly homomorphic encryption scheme provides linear circuit privacy when a PPT simulator $\mathcal{S}$ satisfies*

$$\begin{aligned} & (\text{Eval}_{\text{pk}}(c; a, b), c, \text{sk}, \text{pk}) \\ & \approx_c (\mathcal{S}(\text{sk}, \text{pk}, c, ax + b), c, \text{sk}, \text{pk}) \end{aligned}$$

for every $x, a, b \in R_p$, where the parameters and keys are generated honestly and $c \leftarrow \text{Enc}_{\text{pk}}(x)$.

We use randomized linear evaluation based on the BFV construction of de Castro et al. [8]. For a ciphertext $c \in R_q^2$ and coefficients $a, b \in R_p$, we write $\text{RandLinEval}_p(c; a, b)$ for randomized evaluation returning an encryption of $a \cdot \text{BFV.Decrypt}(c) + b$.

Plaintext Encoding. BFV supports SIMD (single instruction, multiple data) operations, which improve concrete performance by packing many plaintext values into one ciphertext. For a prime plaintext modulus $p \equiv 1 \pmod{2N}$, let $\zeta_1, \dots, \zeta_N \in \mathbb{Z}_p$ be the distinct roots of $X^N + 1$, listed in a fixed order. BFV batching encodes a plaintext vector $\mathbf{m} = (m_1, \dots, m_N) \in \mathbb{Z}_p^N$ as the element of R_p represented by the unique polynomial $y(X) \in \mathbb{Z}_p[X]$ of degree less than N satisfying $y(\zeta_j) = m_j$ for every $j \in [N]$. We say that such a BFV ciphertext has N slots, where each slot stores one value $m_i \in \mathbb{Z}_p$.

3.4 Oblivious Key-Value Store

An oblivious key-value store (OKVS), introduced by Garimella et al. [22], is a randomized data structure for encoding n key-value pairs. Recent OKVS constructions achieve favorable computation and memory complexity [45, 5].

Definition 2 (Oblivious Key-Value Store). *An OKVS is parameterized by a key space $\mathcal{K}$, a value space $\mathcal{V}$, an encoding length ℓ_{kv} , and security parameters λ, κ . We distinguish the public decoding seed $\theta \in \{0, 1\}^\lambda$ from the encoder's private randomness ρ_{enc} . It consists of the following algorithms and properties:*

- **Encode**($L; \theta; \rho_{\text{enc}}$): *Given n key-value pairs $L \in (\mathcal{K} \times \mathcal{V})^n$ with distinct keys, the public seed θ , and private randomness ρ_{enc} , output $P \in \mathcal{V}^{\ell_{\text{kv}}}$ or $\perp$.*
- **Decode**($P, k; \theta$): *Given an encoding P , a query key $k \in \mathcal{K}$, and the public seed θ , deterministically output a value in $\mathcal{V}$.*

We write $P \leftarrow \text{Encode}(L; \theta)$ when fresh private encoding randomness is sampled internally. We use the following properties:

- **Correctness:** *For every successful encoding $P = \text{Encode}(L; \theta; \rho_{\text{enc}}) \neq \perp$, it holds that $\text{Decode}(P, k; \theta) = v$ for every $(k, v) \in L$.*
- **Obliviousness:** *For any two sets of n distinct keys $\{k_i\}_{i \in [n]}$ and $\{k'_i\}_{i \in [n]}$, let $v_1, \dots, v_n \leftarrow_{\S} \mathcal{V}$ be independent uniform values and let $\theta \leftarrow_{\S} \{0, 1\}^\lambda$ be independent of them. For successful encodings, the resulting public views satisfy*

$$\begin{aligned} & (\theta, \text{Encode}(\{(k_i, v_i)\}_{i \in [n]}; \theta)) \\ & \approx_c (\theta, \text{Encode}(\{(k'_i, v_i)\}_{i \in [n]}; \theta)). \end{aligned}$$

- **Double Obliviousness:** For fixed distinct keys and independent uniform values, the successful encoding P is statistically indistinguishable from a uniform element of $\mathcal{V}^{\ell_{kv}}$, even given the public seed θ .
- **Linearity:** Over a finite value field $\mathcal{V}$, a decoding vector $\text{dec}(k; \theta) \in \mathcal{V}^{\ell_{kv}}$ satisfies $\text{Decode}(P, k; \theta) = \langle \text{dec}(k; \theta), P \rangle$.

Lemma 1 (Independent Decoding [23, Appendix A]). Fix distinct programmed keys $k_1, \dots, k_n$ and a query key $k \notin \{k_i\}_{i \in [n]}$, independently of the uniform public seed θ . Assume that the OKVS is linear and that, except with negligible probability, the programmed decoding rows have full row rank and $\text{dec}(k; \theta)$ lies outside their linear span. If encoding failure is negligible and the encoder samples a uniform solution to the programmed equations using private randomness independent of θ and the values, then $\text{Decode}(P, k; \theta)$ is statistically indistinguishable from uniform in $\mathcal{V}$ for arbitrary fixed programmed values.

Lemma 2 (Random Decoding [5]). Let the OKVS be linear and doubly oblivious, with negligible encoding failure. Fix distinct programmed keys and sample their values independently and uniformly from $\mathcal{V}$. For a query key k outside the programmed set and independent of θ , if $\text{dec}(k; \theta) \neq 0$ except with negligible probability, then $\text{Decode}(P, k; \theta)$ is statistically indistinguishable from uniform in $\mathcal{V}$.

3.5 Oblivious Pseudorandom Functions

An oblivious pseudorandom function (OPRF) [17] allows a receiver to evaluate a keyed PRF without revealing its queries to the sender or learning the PRF key. The sender holds a key k , and the receiver obtains $F_k(y)$ for each queried input y . In our protocol descriptions, we suppress the sender’s fixed OPRF key and write $F(\cdot) = F_k(\cdot)$. The sender learns no query value beyond the public batch size, while the receiver learns neither the key nor PRF values at unqueried inputs. We use the batched OPRF framework of Rindal and Schoppmann [47], evaluating one query for each real receiver block, for a public total of N_Y queries.

An oblivious programmable PRF (OPPRF) [30] additionally allows the sender to specify programmed pairs (x_j, v_j) . A query equal to some x_j returns the prescribed value v_j , whereas every nonprogrammed query returns a pseudorandom value. Our block-matching-hint protocol realizes this programming pattern by combining OPRF-derived block keys with an OKVS. We use these OPRF-derived block keys in domain-separated PRF evaluations to derive pads for block-matching hints and block-selective distance sharing. Accordingly, our security analysis uses the $\mathcal{F}_{\text{OPRF}}$ -hybrid model together with the stated OKVS properties.

3.6 Oblivious Transfer

Oblivious transfer (OT) [44] is a fundamental cryptographic primitive in which a sender holds two messages (m_0, m_1) and a receiver holds a choice bit c . The receiver learns m_c without learning m_{1-c} , while the sender learns nothing about c .

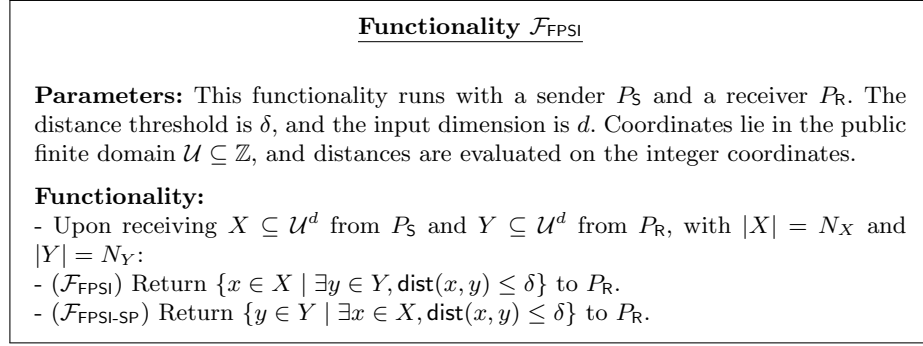


Fig. 4. The ideal functionality of FPSI

In practice, a small number of public-key base OTs are expanded into many OT instances using efficient symmetric-key operations, as in the IKNP OT extension [27]. Our implementation uses the VOLE-style OT of [58], which provides the low-communication OT layer used by the Cheetah 2PC comparison protocol [26].

3.7 Cuckoo and Simple Hashing

Cuckoo hashing [37] maps elements from a large universe into a table of size linear in the input size while supporting efficient lookup. It uses k hash functions $h_1, \dots, h_k : \{0, 1\}^\sigma \rightarrow [1, b]$ to map n elements into $b = \epsilon n$ bins. To insert an entry (k_i, v_i) , the algorithm first tries bin $h_1(k_i)$. If the bin is occupied, it evicts the current entry and relocates it to another candidate bin $h_j(k_{i'})$, where $j \leftarrow [k]$. This eviction process repeats until all entries are placed or a predefined relocation threshold is reached. Cuckoo hashing is often paired with simple hashing. The simple table stores an item in all of its candidate positions, while the Cuckoo table stores it in only one. In our construction, the sender builds the simple table and the receiver builds the Cuckoo table. This pairing aligns a receiver block with the corresponding sender block at the receiver's selected Cuckoo position. For a detailed description of Cuckoo hashing in PSI, see [41, 61].

3.8 Fuzzy PSI

Fuzzy PSI generalizes PSI from exact equality to distance-based matching. Given input sets X from P_S and Y from P_R , standard FPSI returns sender elements that are within distance δ of at least one receiver element. The sender-private variant, FPSI-SP, instead returns receiver elements that are within distance δ of at least one sender element. We define both functionalities in Fig. 3.8.

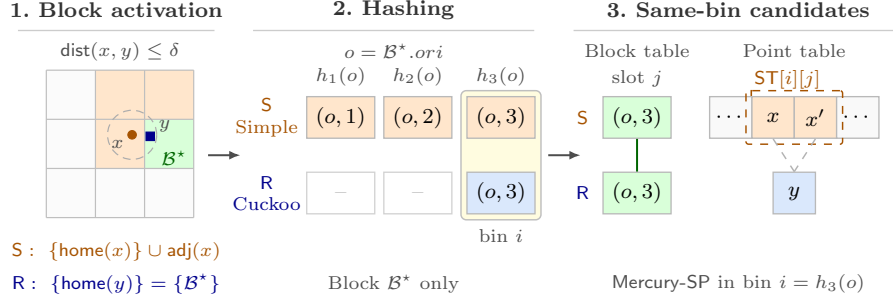


Fig. 5. Spatial hashing for candidate pair generation. For a neighboring pair (x, y) , block activation yields the common active block $\mathcal{B}^* = \text{home}(y)$. On hashing success, sender-side simple hashing and receiver-side Cuckoo hashing place matching blocks in the same bin. In Mercury-SP, the sender segment starting at $\text{ST}[i][j]$ is aligned with the receiver’s unique point entry. Point-table labels denote stored offsets relative to the common block origin o .

4 Our Fuzzy PSI Framework

This section instantiates the framework from Section 2.2 for the sender-private variant Mercury-SP. The standard FPSI variant uses the same geometric filter but a different sender point-table construction and output procedure, as described in Appendix E.

4.1 Spatial Hashing for Candidate Pair Generation

Our candidate-generation procedure builds on the active-cell method of Garimella et al. [23] and the spatial-hashing techniques used in later FPSI constructions [2, 43]. We use spatial hashing to generate candidates for L_2 distance while preserving every neighboring pair when hashing succeeds. As illustrated in Figure 5, block activation followed by sender-side simple hashing and receiver-side Cuckoo hashing produces aligned point tables. These tables provide candidate positions for encrypted distance evaluation and “mask-then-compare”.

Input Blocks and Activation For the integer-coordinate construction, we use a positive integer threshold δ and partition the input universe into grid blocks of side length 2δ . For an origin $\mathbf{a} = (a[1], \dots, a[d])$ with $a[\ell] \in 2\delta\mathbb{Z}$, define

$$\mathcal{B}(\mathbf{a}) := \prod_{\ell=1}^d [a[\ell], a[\ell] + 2\delta).$$

Definition 3 (Input Block). An input block is a grid block $\mathcal{B} = \mathcal{B}(\mathbf{a})$, identified by its origin $\mathcal{B}.\text{ori} = \mathbf{a}$. The origin is a canonical, party-independent identifier of the geometric block and does not depend on the points assigned to that block.

For a point z , its home block $\text{home}(z)$ is the unique block containing z .

Definition 4 (Input Cell Sparsity). *Following the cell-density notation of Piske et al. [43], we define the cell density of an input set $Z \in \{X, Y\}$ as*

$$\rho_c^Z := \max_{\mathcal{B}} |\{z \in Z \mid \text{home}(z) = \mathcal{B}\}|.$$

We call Z cell-sparse if $\rho_c^Z \leq 1$. The construction assumes that both X and Y are cell-sparse.

For a sender point x , write $o = \text{home}(x).\text{ori}$. In coordinate ℓ , the sender activates the home interval and the adjacent interval on the side closer to $x[\ell]$, using the two origins

$$T_\ell(x) := \begin{cases} \{o[\ell], o[\ell] - 2\delta\}, & x[\ell] - o[\ell] < \delta, \\ \{o[\ell], o[\ell] + 2\delta\}, & x[\ell] - o[\ell] \geq \delta. \end{cases}$$

The parties use the activation sets

$$\mathcal{A}_S(x) := \{\mathcal{B}(a) \mid a[\ell] \in T_\ell(x) \text{ for every } \ell\}, \quad \mathcal{A}_R(y) := \{\text{home}(y)\}.$$

We write $\text{adj}(x) = \mathcal{A}_S(x) \setminus \{\text{home}(x)\}$. This coordinatewise rule activates at most 2^d blocks per sender point and conservatively contains every block that can hold an L_2 neighbor, although some activated blocks may not intersect the radius- δ ball.

Definition 5 (Active Block, adapted from [23]). *A block is active for a party if it belongs to one of that party's activation sets, and double-active if it is active for both parties.*

For each party and each of its active blocks $\mathcal{B}$, $\mathcal{B}.\text{pts}$ denotes that party's set of offsets $z - \mathcal{B}.\text{ori}$ for the points z that activate $\mathcal{B}$. For homomorphic evaluation, signed offset coordinates are encoded modulo p . Each receiver-active block contains exactly one receiver offset, since the receiver activates only home blocks and Y is cell-sparse. If a sender point x activates $\mathcal{B}$, then $\text{home}(x)$ differs from $\mathcal{B}$ by at most one grid block in each coordinate. Such sender points can therefore originate from at most 3^d home blocks. Because X is cell-sparse, each of these home blocks contributes at most one sender point, and hence $|\mathcal{B}.\text{pts}| \leq 3^d$.

Definition 6 (Active Point Pair). *A pair $(x, y) \in \text{Pairs}$ is an active point pair if $\text{home}(y) \in \mathcal{A}_S(x)$.*

Lemma 3. *Every neighboring point pair is an active point pair.*

Proof. Let (x, y) be a neighboring pair and set $\mathcal{B} = \text{home}(y)$. Since $\text{dist}(x, y) \leq \delta$, each coordinate satisfies $|x[\ell] - y[\ell]| \leq \delta$. If $x[\ell] - o[\ell] < \delta$, the interval containing $y[\ell]$ has origin $o[\ell]$ or $o[\ell] - 2\delta$. If $x[\ell] - o[\ell] \geq \delta$, its origin is $o[\ell]$ or $o[\ell] + 2\delta$. Thus, $\mathcal{B}.\text{ori}[\ell] \in T_\ell(x)$ in every coordinate, and the sender activates $\mathcal{B}$. The receiver also activates $\mathcal{B}$, so (x, y) is an active point pair.

Thus, activation preserves every neighboring pair, but active point pairs may still have distance greater than δ .

Generating Active Block Sets The sender and receiver generate their active block sets by applying the activation rules independently to their own inputs. For the receiver, the active block set is

$$Y^{(A)} = \{\text{home}(y) \mid y \in Y\},$$

and the cell sparsity of Y gives $|Y^{(A)}| = N_Y$. Each $\text{home}(y).pts$ contains the unique offset $y - \text{home}(y).ori$. For the sender, each $x \in X$ contributes the offset $x - \mathcal{B}.ori$ to every block $\mathcal{B} \in \mathcal{A}_S(x)$. Algorithm 1 gives the complete sender-side construction of $X^{(A)}$. Block records are indexed by their origins, with each record storing the offsets of all sender points that activate the corresponding block.

Algorithm 1 Sender-side active-block construction

Require: Sender input set X

Ensure: Active block set $X^{(A)}$ with point-offset sets

```

1:  $X^{(A)} \leftarrow \emptyset$ 
2: for  $x \in X$  do
3:   for  $\mathcal{B} \in \mathcal{A}_S(x)$  do
4:     if  $\mathcal{B} \notin X^{(A)}$  then
5:       initialize  $\mathcal{B}.pts \leftarrow \emptyset$ 
6:        $X^{(A)} \leftarrow X^{(A)} \cup \{\mathcal{B}\}$ 
7:        $\mathcal{B}.pts \leftarrow \mathcal{B}.pts \cup \{x - \mathcal{B}.ori\}$ 
8: return  $X^{(A)}$ 

```

Since each sender point activates at most 2^d blocks, this construction creates at most $2^d N_X$ point-offset entries before hashing.

Hash-Table Construction The hash-table construction turns active blocks into aligned bins. The receiver uses Cuckoo hashing, whereas the sender uses simple hashing. The parties agree on a common table size m and three public hash functions

$$h_1, h_2, h_3 : 2\delta\mathbb{Z}^d \longrightarrow [m].$$

For each block $\mathcal{B}$, define its candidate-bin set as

$$\text{Cand}(\mathcal{B}) := \{h_t(\mathcal{B}.ori) \mid t \in [3]\} \subseteq [m].$$

Because the table size and hash functions are public, both parties derive the same $\text{Cand}(\mathcal{B})$ locally from $\mathcal{B}.ori$.

In the hash tables, a block $\mathcal{B}$ placed using h_t is identified by $\mathcal{B}.id = (\mathcal{B}.ori, t)$, where $t \in [3]$ is the hash-function index. A sender block matches a receiver block if they have the same origin and hash-function index.

We use C_{pt} as the public padded capacity of each sender point-table bin and C_{act} as the length of the sender window processed by subsequent threshold testing. The parameters satisfy $3^d \leq C_{\text{act}} \leq C_{\text{pt}}$, ensuring that this window can

cover every real point of a matching sender block. The table parameters are selected as discussed in Section 5.1.

Receiver table. The receiver inserts each $\mathcal{B} \in Y^{(A)}$ using standard Cuckoo hashing [41]. The Cuckoo procedure chooses a hash-function index $t \in [3]$ and stores the block identified by $(\mathcal{B}.ori, t)$ in bin $i = h_t(\mathcal{B}.ori)$ of BT_R . The unique corresponding point offset is stored in $\text{PT}_R[i]$. On successful insertion, each receiver bin contains one real block and its unique point offset or is filled with dummy entries $\perp$.

Sender table. The sender applies simple hashing using the same public hash functions h_1, h_2, h_3 as the receiver. For every $\mathcal{B} \in X^{(A)}$ and every hash-function index $t \in [3]$, the sender stores the block with identifier $(\mathcal{B}.ori, t)$ in bin $h_t(\mathcal{B}.ori)$ of BT_S . Blocks with different hash-function indices are retained even when they occupy the same bin. The point offsets of the block $\text{BT}_S[i][j]$ are stored in an arbitrary order as a contiguous segment of $\text{PT}_S[i]$, starting at $\text{ST}[i][j]$. We call $\text{ST}[i][j]$ the segment-start index, reserving point offset for a vector of the form $z - \mathcal{B}.ori$. The sender aborts if any bin contains more than C_{pt} point entries. Otherwise, it pads each bin to C_{pt} entries, applies an independently and uniformly sampled cyclic rotation, and updates the segment-start indices. Segments and candidate windows are read consecutively from their start indices, wrapping around at the end of the bin. This contiguous, cyclic layout is used by **Mercury-SP**. The standard variant instead permutes all real and dummy positions, as specified in Appendix E. The active-point validity shares generated in Section 4.3 hide the actual length of each segment.

Hashing success and block alignment. Successful hashing requires two conditions: every receiver-active block is placed, and the complete load of every sender bin fits in its point table. For each sender bin $i \in [m]$, define the point load

$$L_i := \sum_{\mathcal{B} \in X^{(A)}} \sum_{t \in [3]} \mathbf{1}[h_t(\mathcal{B}.ori) = i] |\mathcal{B}.pts|.$$

Let Succ_{ck} be the event that receiver-side Cuckoo hashing places every $\mathcal{B} \in Y^{(A)}$ in some bin $i \in \text{Cand}(\mathcal{B})$. Let

$$\text{Succ}_{\text{bin}} := \left[\max_{i \in [m]} L_i \leq C_{\text{pt}} \right]$$

be the event that no sender point-table bin overflows. We define the hashing-success event as

$$\text{Succ}_{\text{hash}} := \text{Succ}_{\text{ck}} \wedge \text{Succ}_{\text{bin}}.$$

If either hashing-success condition fails, the protocol aborts rather than truncating a sender point segment.

We call (x, y) a *same-bin point pair* if the table entries representing x and y occur in $\text{PT}_S[i]$ and $\text{PT}_R[i]$, respectively, for some $i \in [m]$. The following theorem states the stronger block-alignment invariant guaranteed by successful spatial hashing.

Theorem 1 (Completeness of Spatial Hashing). *Suppose that both X and Y are cell-sparse as in Definition 4. If $\text{Succ}_{\text{hash}}$ occurs, then for every neighboring point pair (x, y) , with $\mathcal{B} = \text{home}(y)$, there exist a hash-function index t , a bin $i = h_t(\mathcal{B}.ori)$, and a sender slot j such that $\text{BT}_R[i]$ and $\text{BT}_S[i][j]$ both have identifier $(\mathcal{B}.ori, t)$. The receiver entry satisfies $\text{PT}_R[i] = y - \mathcal{B}.ori$. In Mercury-SP, the cyclic sender segment beginning at $\text{ST}[i][j]$ contains $x - \mathcal{B}.ori$. In particular, (x, y) is a same-bin point pair.*

Proof. Fix an arbitrary $(x, y) \in \text{Pairs}$ satisfying $\text{dist}(x, y) \leq \delta$, and set $\mathcal{B} = \text{home}(y)$. By Lemma 3, $\mathcal{B} \in X^{(A)} \cap Y^{(A)}$, and the sender-side set $\mathcal{B}.pts$ contains the offset $x - \mathcal{B}.ori$. Because Succ_{ck} holds, the receiver places the block with identifier $(\mathcal{B}.ori, t)$ in bin $i = h_t(\mathcal{B}.ori)$ for some chosen hash-function index t . The receiver-table construction therefore stores an entry identifying $\mathcal{B}$ in $\text{BT}_R[i]$ and stores $y - \mathcal{B}.ori$ in $\text{PT}_R[i]$. Sender-side simple hashing inserts a block with the same origin and hash-function index into bin i , where it occupies some sender slot j . The event Succ_{bin} ensures that the complete segment $\mathcal{B}.pts$, including the offset of x , is retained in $\text{PT}_S[i]$. The subsequent cyclic rotation preserves this segment and moves its start to the updated index $\text{ST}[i][j]$. Thus, the entries for x and y are aligned through matching blocks in bin i , making (x, y) a same-bin point pair.

Result of spatial hashing. On $\text{Succ}_{\text{hash}}$, the aligned point tables contain one receiver slot and C_{pt} sender slots per bin, including padding. Encrypted distance evaluation processes these mC_{pt} logical positions using the stored point offsets. For matching blocks, both offsets use the same origin, so their squared L_2 distance equals that of the original point pair. The subsequent “mask-then-compare” stage identifies neighboring point pairs among these candidates without revealing their distances.

4.2 Batched Encrypted Distance Evaluation

We use BFV batching to compute encrypted distance expressions for the same-bin candidates produced by spatial hashing [52]. Figure 6 illustrates how the point tables are packed, with each row representing one bin containing C_{pt} sender point slots and one receiver point slot. For each point slot and coordinate, we pack the coordinate values across bins into a vector, so corresponding plaintext slots on both sides represent the same bin. This alignment allows SIMD homomorphic operations to process the same candidate position across bins in parallel. The packing produces $C_{\text{pt}} \cdot d$ logical sender coordinate vectors, collectively denoted by pt_S , and d logical receiver coordinate vectors, collectively denoted by pt_R .

The receiver encrypts pt_R and sends the resulting ciphertext collection ct_R to the sender. To protect the evaluated values, the sender incorporates the slot masks mask_{out} constructed by block-selective distance sharing in Section 4.3. For each sender position α , we write $\text{mask}_{\text{out}}[\alpha]$ for the vector of masks $\text{mask}_{\text{out}}[i][\alpha]$

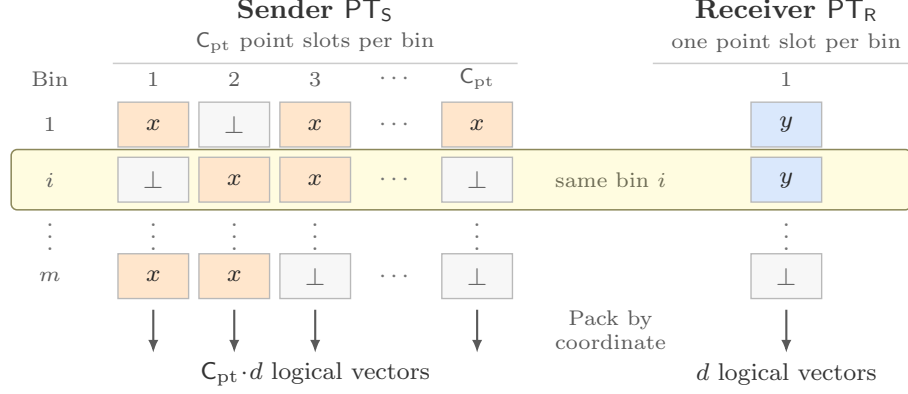


Fig. 6. Packed point-table layout for batched distance evaluation. Each row represents one spatial-hashing bin, and the highlighted band illustrates the same-bin alignment. The labels x, y represent stored point offsets, and $\perp$ denotes padding. Coordinate-wise packing gives $C_{pt} \cdot d$ sender logical vectors and d receiver logical vectors. Several logical vectors may share one BFV plaintext without changing this alignment.

Algorithm 2 Batched Encrypted Distance Evaluation with Masking

Require: Receiver ciphertexts $\{\text{ct}_R[k]\}_{k \in [d]}$
Require: Sender plaintexts $\{\text{pt}_S[\alpha][k]\}_{\alpha \in [C_{pt}], k \in [d]}$
Require: Packed slot masks $\{\text{mask}_{out}[\alpha]\}_{\alpha \in [C_{pt}]}$
Ensure: Masked distance ciphertexts $\{\text{ct}_{dis}[\alpha]\}_{\alpha \in [C_{pt}]}$

- 1: **for** $\alpha \in [C_{pt}]$ **do**
- 2: $b_\alpha \leftarrow \sum_{k \in [d]} \text{pt}_S[\alpha][k]^2 + \text{mask}_{out}[\alpha]$
- 3: $\text{ct}_{dis}[\alpha] \leftarrow \text{RandLinEval}_p(\text{ct}_R[1]; -2\text{pt}_S[\alpha][1], b_\alpha)$
- 4: **for** $k = 2, \dots, d$ **do**
- 5: $\text{ct}[\alpha][k] \leftarrow \text{RandLinEval}_p(\text{ct}_R[k]; -2\text{pt}_S[\alpha][k], 0)$
- 6: $\text{ct}_{dis}[\alpha] \leftarrow \text{ct}_{dis}[\alpha] + \text{ct}[\alpha][k]$
- 7: **return** $\{\text{ct}_{dis}[\alpha]\}_{\alpha \in [C_{pt}]}$

across bins i . Algorithm 2 evaluates the encrypted distance expressions with these masks included in the randomized evaluation.

For real entries in bin i and sender position α , write $\tilde{x} = \text{PT}_S[i][\alpha]$ and $\tilde{y} = \text{PT}_R[i]$ for the stored point offsets. The plaintext value in slot i of $\text{ct}_{dis}[\alpha]$ is

$$\sum_{\ell=1}^d \tilde{x}[\ell]^2 - 2 \sum_{\ell=1}^d \tilde{x}[\ell] \tilde{y}[\ell] + \text{mask}_{out}[i][\alpha] \quad \text{in } \mathbb{F}_p.$$

Because $\sum_{\ell=1}^d \tilde{y}[\ell]^2$ depends only on the receiver's stored offset, it need not be evaluated homomorphically. After decrypting the response, the receiver adds this term to obtain $\text{Dist}(\tilde{x}, \tilde{y}) + \text{mask}_{out}[i][\alpha]$ in $\mathbb{F}_p$. For matching blocks, the two offsets share the same origin, so $\text{Dist}(\tilde{x}, \tilde{y}) = \text{Dist}(x, y)$ for the represented original points x, y .

Functionality $\mathcal{F}_{\text{BMH}}$

Parameters: The functionality for Mercury-SP uses m bins with cyclic sender segments and $3^d \leq C_{\text{act}} \leq C_{\text{pt}}$.

Inputs: The sender provides BT_S and the segment-start table ST . The receiver provides BT_R .

Functionality: For each bin $i \in [m]$, proceed as follows. If $\text{BT}_R[i] \neq \perp$ and a matching sender block exists, let j_i^* be the unique index satisfying

$$\text{BT}_S[j_i^*].id = \text{BT}_R[i].id$$

and set $\ell_i = |\text{BT}_S[j_i^*].pts|$. Otherwise, set $j_i^* = \perp$ and $\ell_i = 0$.

- *Match indicator:* Set $eq[i] = \mathbf{1}[j_i^* \neq \perp]$ and return Boolean shares $\langle eq[i] \rangle_{S,R}$.
- *Segment-start index:* If $eq[i] = 1$, return $\text{ST}_R[i] = \text{ST}[j_i^*]$ to P_R . For a real but nonmatching receiver block, return a uniform $\text{ST}_R[i] \in [C_{\text{pt}}]$. For an empty receiver bin, return the local default $\text{ST}_R[i] = 1$.
- *Position validity:* For each $k \in [C_{\text{act}}]$, set

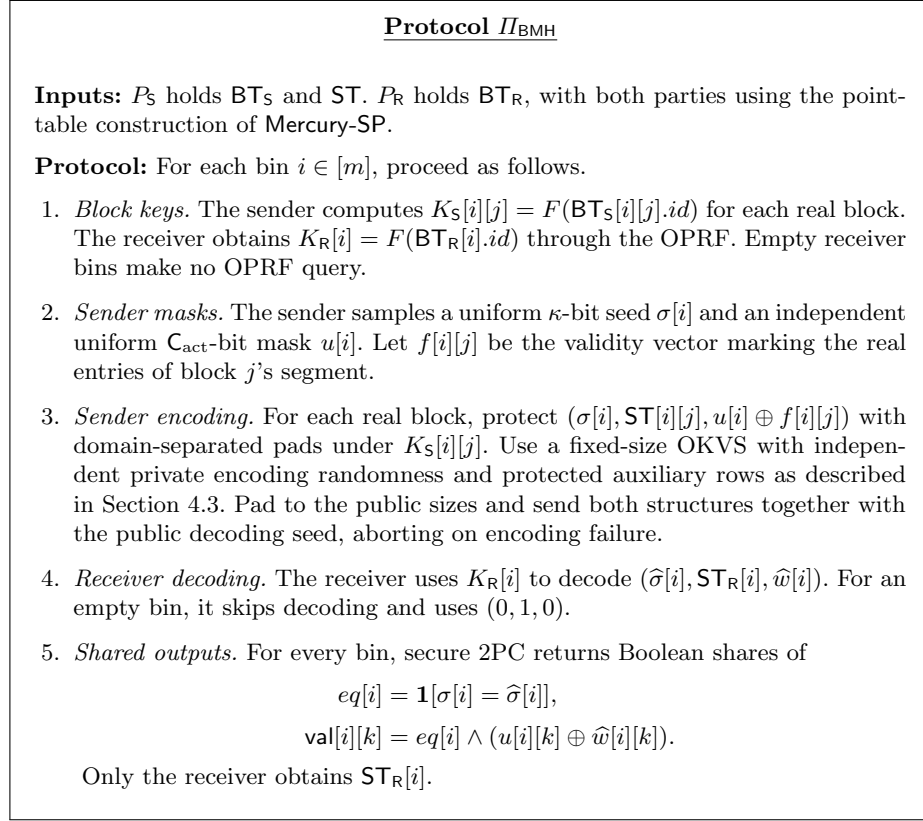
$$\text{val}[i][k] = eq[i] \wedge \mathbf{1}[k \leq \ell_i]$$

and return Boolean shares $\langle \text{val}[i][k] \rangle_{S,R}$.

Fig. 7. Ideal functionality for block-matching hints. For each bin, it returns a secret-shared match indicator and position-validity vector, together with a segment-start index that is correct only for a matching block.

4.3 Mask-Then-Compare

Spatial hashing reduces the search space to same-bin candidates, but bin alignment alone does not establish proximity because hash collisions and padding may introduce structurally invalid candidates, while points assigned to the same active block may still be farther apart than δ . The “mask-then-compare” stage removes both classes of false positives by applying the following three-step procedure without revealing the candidate distances. First, block-matching hints privately determine whether a sender block matches the nonempty receiver block by comparing their identifiers. The resulting segment-start index selects a window of C_{act} entries in $\text{PT}_S[i]$, which we call candidate positions. The hints secret-share a validity bit for each position, marking it valid only when the block matches and the position contains a real sender point. Next, block-selective distance sharing makes the share difference equal the squared distance at valid positions and pseudorandom over $\mathbb{F}_p$ at invalid positions. Finally, the secure masked-range test compares each shared squared distance with the public squared threshold δ^2 , and the 2PC circuit combines the comparison result with its validity bit so that invalid positions cannot affect the output.

**Fig. 8.** Block-matching hints for **Mercury-SP**.

Block-Matching Hints The asymmetric table construction in Section 4.1 places at most one real active block in each receiver bin, while the corresponding sender bin may contain several active blocks. The subsequent stages must therefore select the sender segment associated with the receiver block and exclude its padded positions without disclosing the match indicator. Figure 7 formalizes this task through three outputs for each bin: a secret-shared match indicator $eq[i]$, a segment-start index $\text{ST}_R[i]$, and secret-shared position-validity bits $\text{val}[i][k]$. These outputs provide a privacy-preserving description of the sender segment corresponding to a double active block. When the double active block exists, the index identifies the corresponding segment, and the secret-shared validity bits mark its real entries. For a nonmatching real block, all validity bits are zero and the receiver obtains a random index. An empty receiver bin uses fixed local decoding defaults but participates in the same 2PC computation, yielding shared zero validity bits outside the budgeted false-equality event.

We realize this interface using an OPRF, an OKVS [47], and protected auxiliary rows. The key $F(\mathcal{B}.id)$ binds each block to its geometric origin and hash-

function index. Separate PRF domains derive the OKVS index, the hint pad, the auxiliary-row pads, and the distance pads from the full OPRF key. The public derivation context binds the session, the bin, and the relevant row or candidate index. Each bin uses an OKVS padded to C_{pt} key-value pairs and fixed-size auxiliary data to hide its real occupancy. For a matching block, the receiver uses its block key to recover the segment-start index and a masked validity vector. For a nonmatching block, decoding yields pseudorandom values in the same domains. Figure 8 gives the protocol, with encoding details in Appendix C.1.

Each bin uses one random match seed $\sigma[i]$ and one uniform validity mask $u[i]$. For sender block j in bin i , let $f[i][j] \in \{0, 1\}^{C_{act}}$ indicate its real segment entries, with $f[i][j][k] = 1$ if $k \leq |BT_S[i][j].pts|$ and 0 otherwise. Let $\hat{w}[i] \in \{0, 1\}^{C_{act}}$ denote the decoded validity vector held by the receiver. For a matching block j , the receiver recovers the seed $\sigma[i]$ and obtains $\hat{w}[i] = u[i] \oplus f[i][j]$. The parties compare the seeds and compute $val[i][k] = eq[i] \wedge (u[i][k] \oplus \hat{w}[i][k])$ within secure 2PC, revealing only shares of the resulting bits.

Theorem 2. *Protocol Π_{BMH} realizes the functionality $\mathcal{F}_{BMH}$ used in Mercury-SP against static semi-honest adversaries under the OPRF, PRF, OKVS, and 2PC in Appendix C, assuming negligible sampling errors and failure probabilities.*

Proof (Sketch). OPRF consistency supplies the same key for matching blocks. A matching lookup yields the prescribed start index and a uniformly masked validity vector, whereas a nonmatching lookup yields a random legal index. The proof simulates the complete OKVS encoding and all auxiliary rows consistently with every receiver-known key, not only the selected lookup. Secure 2PC hides the match and validity values, except for a false seed equality with total probability at most $q_{eq}2^{-\kappa}$, where $q_{eq} \leq m$. Appendix C gives the full proof.

Block-Selective Distance Sharing The block-matching stage provides the receiver with a segment-start index for each bin, while the parties obtain secret shares of the validity bits for the C_{act} positions beginning at that index. When a matching sender block exists, the index locates its point segment and the validity bits identify the real entries in that segment. Block-selective distance sharing uses the segment-start index to select the corresponding C_{act} encrypted distance values in each bin and converts them into additive shares. At a valid position, the receiver’s share minus the sender’s share equals the squared distance. At an invalid position, this difference is pseudorandom over $\mathbb{F}_p$, with the concrete PRF and sampling errors accounted for separately.

The concrete construction supplies the masks for the encrypted evaluation in Section 4.2 and uses the Mercury-SP alignment information to recover the selected shares. Let $D_K(i, k) \in \mathbb{F}_p$ denote a distance-domain PRF pad under the full OPRF key K , with the current session context implicit. For each bin, the sender samples $mask_{in}[i][k] \leftarrow_{\$} \mathbb{F}_p$ for $k \in [C_{act}]$ and reuses this vector across the bin’s blocks. The k -th real point of block j receives the combined mask

$$mask_{in}[i][k] + D_{K_S[i][j]}(i, k), \quad K_S[i][j] = F(BT_S[i][j].id).$$

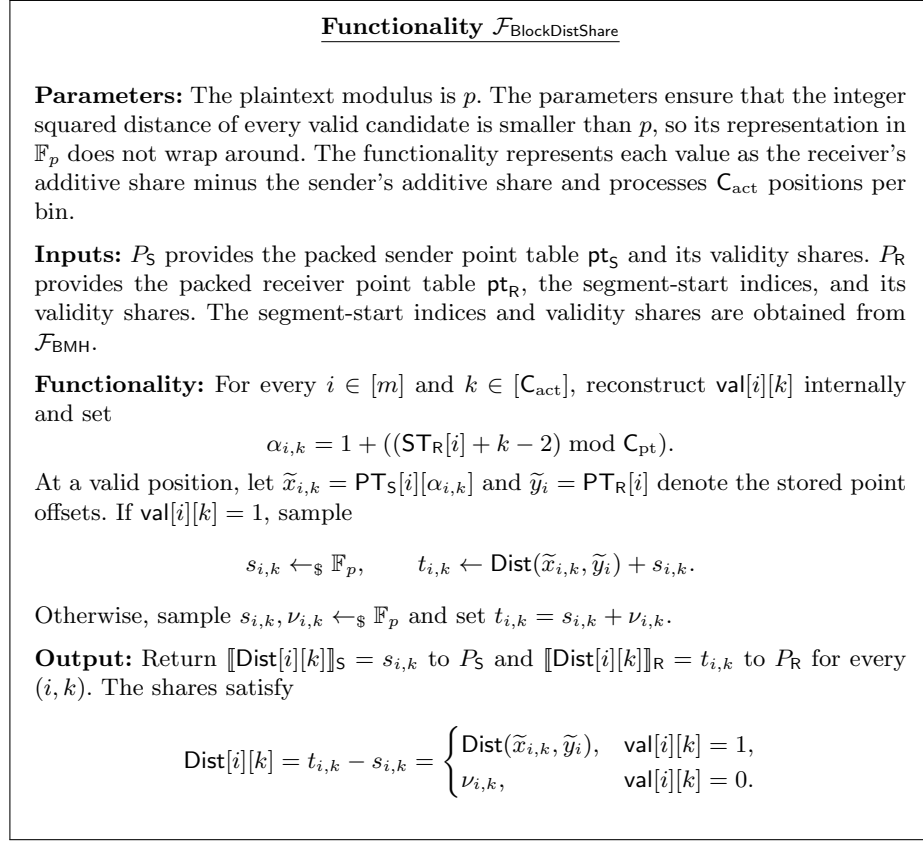


Fig. 9. Ideal interface for block-selective distance sharing in Mercury-SP after block matching.

Each real slot is written only by its owning block, and all dummy slots receive independent uniform masks. The sender incorporates these masks into the affine constants of Algorithm 2, so its output is already masked. After decrypting the entire response and adding its squared-norm term, the receiver extracts the k -th position of its cyclic window and subtracts $D_{K_R[i]}(i, k)$. Matching keys and candidate indices cancel the distance pads, while the sender's distance-protection share remains unknown to the receiver. Figures 10 and 11 respectively specify the protocol and illustrate its two possible outcomes.

Theorem 3. *Protocol $\Pi_{\text{BlockDistShare}}$ realizes the interface for Mercury-SP specified in Fig. 9 against static semi-honest adversaries under the OPRF, PRF, BFV input-privacy, and circuit-privacy requirements in Appendix C.*

Proof (Sketch). BFV input privacy hides the receiver's encrypted table from the sender. At a valid position, matching keys and candidate indices cancel the

Protocol $\Pi_{\text{BlockDistShare}}$

Inputs: P_S holds $\text{pt}_S, \text{BT}_S, \text{ST}$, and P_R holds $\text{pt}_R, \text{BT}_R, \text{ST}_R$. The parties retain the block keys and validity shares from Π_{BMH} . Let $D_K(i, k) \in \mathbb{F}_p$ denote the session-bound distance-domain PRF pad under the full block key K .

Protocol:

1. *Encrypted query.* The receiver sends BFV encryptions of its d packed coordinate vectors, using its secret key sk .
2. *Slot masks.* For each bin i , the sender samples independent uniform masks $\text{mask}_{in}[i][k]$ and $\text{mask}_{out}[i][\alpha]$ over $\mathbb{F}_p$, for $k \in [\text{C}_{act}]$ and $\alpha \in [\text{C}_{pt}]$. For each real point k of block j , it sets

$$\begin{aligned} \text{pos}_{i,j}(k) &= 1 + ((\text{ST}[i][j] + k - 2) \bmod \text{C}_{pt}), \\ \text{mask}_{out}[i][\text{pos}_{i,j}(k)] &= \text{mask}_{in}[i][k] + D_{K_S[i][j]}(i, k). \end{aligned}$$

Only the owning block writes each real slot, and dummy masks remain unchanged.

3. *Masked evaluation.* The sender runs Algorithm 2 on the query, pt_S , and mask_{out} packed as in Fig. 6. It returns the masked response with circuit privacy as in Appendix C.2.
4. *Share recovery.* The receiver decrypts the entire response and adds its squared-norm term slotwise to obtain $V[i][\alpha]$. For each bin i and candidate $k \in [\text{C}_{act}]$, it computes

$$\begin{aligned} \alpha_{i,k} &= 1 + ((\text{ST}_R[i] + k - 2) \bmod \text{C}_{pt}), \\ \llbracket \text{Dist}[i][k] \rrbracket_R &= V[i][\alpha_{i,k}] - D_{K_R[i]}(i, k). \end{aligned}$$

Empty bins use $\text{ST}_R[i] = 1$ and a zero distance pad.

Output: The sender outputs $\llbracket \text{Dist}[i][k] \rrbracket_S = \text{mask}_{in}[i][k]$, and the receiver outputs the shares above.

Fig. 10. Protocol for block-selective distance sharing in Mercury-SP.

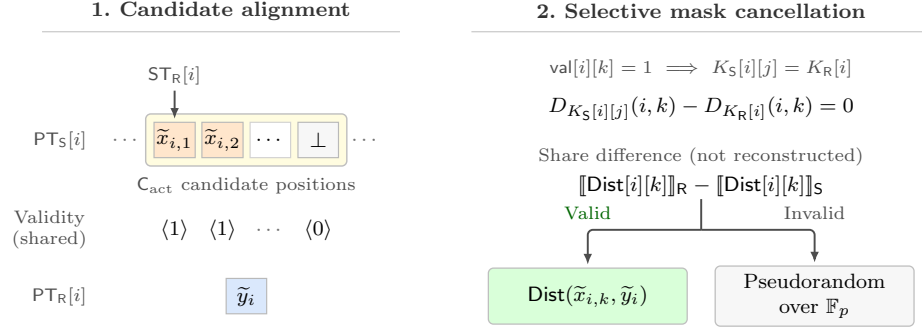


Fig. 11. Illustration of block-selective distance sharing in Mercury-SP. The matching-block example shows the selected candidate window and shared validity bits. The labels $\tilde{x}_{i,k}, \tilde{y}_i$ denote stored point offsets. Valid positions yield squared distances in shared form, while invalid positions retain pseudorandom share differences.

distance pads, leaving the true squared distance in shared form. Every other real slot is protected by an unknown owner key, and dummy slots have independent uniform masks. The proof simulates the complete vector of decrypted slots while retaining the per-bin reuse of $mask_{in}[i]$, then invokes circuit privacy for the full response batch against a receiver holding the decryption key and the randomness used to encrypt the query.

Secure Masked-Range Test After block-selective distance sharing, the two parties hold additive shares for each candidate position. They must test these shares against δ^2 without reconstructing the squared distance. We define this test over the plaintext field of BFV scheme.

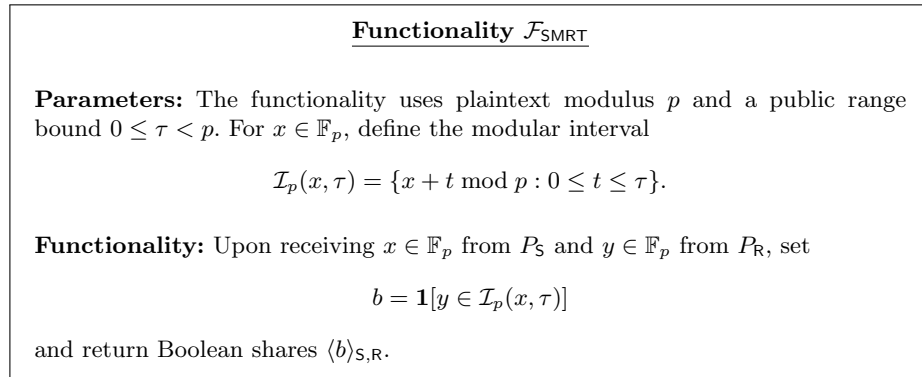


Fig. 12. Ideal functionality for the secure masked-range test.

For candidate position (i, k) , the sender inputs $\llbracket \text{Dist}[i][k] \rrbracket_S$ and the receiver inputs $\llbracket \text{Dist}[i][k] \rrbracket_R$ to $\mathcal{F}_{\text{SMRT}}$ with $\tau = \delta^2$, obtaining Boolean shares of $b[i][k]$. The final candidate predicate is

$$\text{near}[i][k] = b[i][k] \wedge \text{val}[i][k], \quad \text{Ins}[i] = \bigvee_{k \in [C_{\text{act}}]} \text{near}[i][k].$$

By the output relation in Fig. 8, $\text{val}[i][k] = \text{eq}[i] \wedge f[i][j][k]$, so the match indicator is already incorporated into the validity value. Consequently, a bin contributes to the output only when it contains a matching active block and a neighboring point pair. The implementation of $\mathcal{F}_{\text{SMRT}}$ uses the secure comparison primitive described in Appendix B.

4.4 FPSI-SP Protocol

The components above compose into the **Mercury-SP** protocol in Fig. 13. Spatial hashing produces aligned candidate tables, after which block-matching hints privately locate the matching sender segment when one exists and secret-share the validity of its positions. The concrete $\Pi_{\text{BlockDistShare}}$ protocol integrates batched encrypted distance evaluation with selective masking to produce hidden distance shares for the selected positions. The parties apply $\mathcal{F}_{\text{SMRT}}$ to the distance shares and securely compute $\text{near}[i][k] = b[i][k] \wedge \text{val}[i][k]$ from the resulting comparison and validity shares. Since $\text{val}[i][k] = 1$ only for a real point in the matching sender block, $\text{near}[i][k]$ can be one only for a valid candidate that satisfies the distance threshold.

The receiver learns only the per-bin disjunction of the candidate predicates and returns the corresponding points of Y . For standard FPSI, returning sender points also exposes their selected candidate positions. Appendix E therefore gives a separate construction that uniformly permutes all sender-bin slots, replaces cyclic windows with fixed-size address sets, and retrieves the selected points by OT. Theorem 7 establishes its security by simulating the receiver’s view of the exposed positions and returned points.

Correctness We discuss the **Mercury-SP** construction here and analyze the different Non-SP version and output procedure in Appendix E. We discuss correctness for honest executions on inputs X and Y that satisfy Definition 4, with $C_{\text{act}} \geq 3^d$. The plaintext modulus p is chosen to exceed the integer squared distance at every candidate position for which $\text{val}[i][k] = 1$. In a matching block, sender offsets lie coordinatewise in $[-\delta, 3\delta)$, while receiver offsets lie in $[0, 2\delta)$. For integer coordinates and integer δ , each coordinate difference therefore has magnitude at most $3\delta - 1$, so $p > d(3\delta - 1)^2$ is sufficient. The completeness argument applies when $\text{Succ}_{\text{hash}}$ occurs and the invoked subprotocols complete successfully.

Consider a neighboring point pair (x, y) with $\text{dist}(x, y) \leq \delta$, or equivalently $\text{Dist}(x, y) \leq \delta^2$. Theorem 1 guarantees that the block $\mathcal{B} = \text{home}(y)$ appears

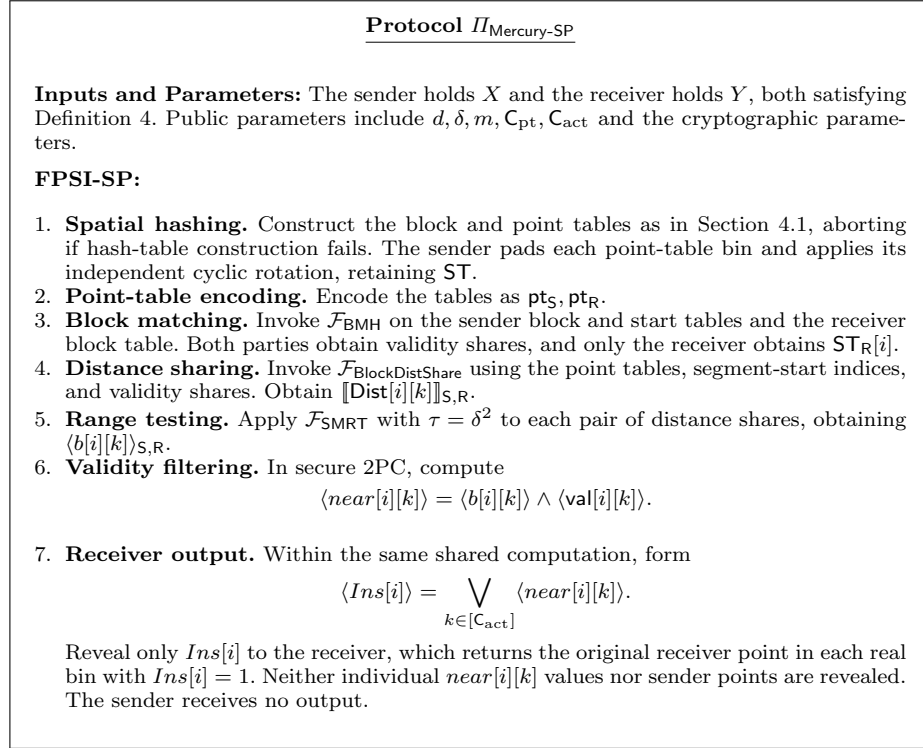


Fig. 13. The Mercury-SP protocol. Its cyclic candidate windows support shared distance testing and one membership output per receiver bin. Standard FPSI uses the separate construction in Appendix E.

as matching sender and receiver block entries in some bin i , and that the corresponding sender segment contains the offset $x - \mathcal{B}.ori$. The sender segment has length at most $3^d \leq C_{act}$, so every real position in the segment lies within the block-matching window. Functionality $\mathcal{F}_{BMH}$ therefore returns $eq[i] = 1$, the correct segment-start index, and $val[i][k] = 1$ for the position k representing x within the segment. The correctness of $\mathcal{F}_{BlockDistShare}$ makes the share difference equal to $\text{Dist}(\tilde{x}_{i,k}, \tilde{y}_i) = \text{Dist}(x, y)$, since both offsets use the origin $\mathcal{B}.ori$. The range test instantiated with $\tau = \delta^2$ returns $b[i][k] = 1$. Hence $near[i][k] = 1$, which makes $Ins[i] = 1$ and causes FPSI-SP to output y . Thus, every receiver point prescribed by $\mathcal{F}_{FPSI-SP}$ appears in the protocol output.

Suppose that FPSI-SP outputs a receiver point y stored in bin i . The output rule gives $Ins[i] = 1$, so there exists a candidate position $k \in [C_{act}]$ such that $near[i][k] = 1$. Because $near[i][k] = b[i][k] \wedge val[i][k]$, both $b[i][k]$ and $val[i][k]$ equal one. The condition $val[i][k] = 1$ identifies a real sender point x in the block matching the receiver block in bin i . The distance-share relation therefore equals $\text{Dist}(x, y)$, and $b[i][k] = 1$ implies $\text{Dist}(x, y) \leq \delta^2$. Equivalently, $\text{dist}(x, y) \leq \delta$, so y belongs to the output of $\mathcal{F}_{FPSI-SP}$.

If receiver-side Cuckoo hashing fails or a sender-bin load exceeds C_{pt} , then Succ_{hash} does not hold and the protocol aborts rather than using incomplete tables. The block-matching protocol likewise aborts if OKVS encoding fails. Outside identifier collisions, false match-seed equalities, and HE or secure-computation correctness errors, every non-aborting execution returns exactly the output prescribed by $\mathcal{F}_{FPSI-SP}$.

Security Proof The security claim is for static semi-honest adversaries and the sender privacy output functionality. We analyze successful table construction and assume negligible sampling errors and failure probabilities, as in Appendix C.3.

Theorem 4. *On cell-sparse inputs and successful table construction, $\Pi_{\text{Mercury-SP}}$ realizes $\mathcal{F}_{FPSI-SP}$ against static semi-honest adversaries under the cryptographic requirements of Appendix C, including circuit privacy, up to the stated error terms.*

Proof (Sketch). The sender's view is simulated using OPRF query privacy, BFV input privacy, and secure 2PC, since it receives no output. For a corrupted receiver, independent cyclic rotations make the selected start indices uniform, and the validity words and distance values remain masked. A simulator generates the complete hint tables and masked response consistently with the receiver's block keys. Only the per-bin disjunction is opened, and the ideal output determines that bit for every receiver bin. Composing the secure computations with the circuit-privacy simulator therefore reveals no additional information. Appendix C.3 gives the full proof, while Appendix E separately simulates the larger Non-SP view.

5 Performance Evaluation

We evaluate the runtime and communication of both Mercury variants under the L_2 metric across balanced and unbalanced inputs.

5.1 Parameter Selection

We use the following BFV and hash-table parameters.

BFV Parameters We instantiate BFV with 128-bit security following [1]. The number of ciphertext slots is set to $N = 8192$ or $N = 16384$, depending on the receiver-set size, to reduce unused slots. We use randomized linear evaluation based on [8], subject to the circuit-privacy premise in Appendix C.2. The coefficient-modulus chain has prime bit lengths $\{30, 56, 56\}$, with 86 bits at the initial evaluation level and 30 bits in each returned distance ciphertext. The Gaussian standard deviations are $\sigma_{\text{code}} = 2^{23}$ and $\tau_{\text{code}} = 2^{50}$. We use a 20-bit plaintext modulus p for $\delta \in \{32, 64\}$ and a 21-bit modulus for the evaluated $\delta = 128$ configuration, which has $d = 2$ and $N = 8192$. The chosen primes satisfy $p > d(3\delta - 1)^2$ in the evaluated configurations, preventing modular wraparound of every valid candidate squared distance as shown in Section 4.4.

Hash Table Parameters Let $\epsilon_{\text{ck}} = \Pr[\neg \text{Succ}_{\text{ck}}]$ and $\epsilon_{\text{bin}} = \Pr[\neg \text{Succ}_{\text{bin}}]$ denote the receiver-side Cuckoo insertion-failure probability and the sender-bin overflow probability, respectively. The hashing-success conditions in Section 4.1 and the union bound give

$$\Pr[\neg \text{Succ}_{\text{hash}}] \leq \epsilon_{\text{ck}} + \epsilon_{\text{bin}}.$$

For receiver-side Cuckoo hashing, we follow the parameter-selection analysis of Pinkas et al. [41], targeting a negligible insertion-failure probability. The cell sparsity of Y gives $|Y^{(A)}| = N_Y$. Concretely, we use three hash functions and a common table size $m \approx 1.27N_Y$, following their recommended parameters for a target Cuckoo failure probability of 2^{-40} . We estimate the maximum bin occupancy of the sender’s point table PT_{S} . Because each sender-active block may contain multiple points, with $|\mathcal{B}.pts| \in [1, C_{\text{act}}]$, the standard balls-into-bins model does not accurately describe the point-table occupancy. We therefore use the weighted balls-and-bins model [4, 15] to choose C_{pt} large enough that the overflow probability is negligible for the target setting. The resulting capacity is fixed throughout the comparison phase. Under the cell sparsity of X , the geometric activation rule gives $|\mathcal{B}.pts| \leq 3^d$. We therefore choose $C_{\text{act}} \geq 3^d$, making the per-block capacity a deterministic consequence of the input assumption rather than a separate admissibility condition. The implementation checks this bound and the realized bin loads before running the comparison phase and aborts on a violation. It does not silently discard excess points.

5.2 Implementation Details

We implement both Mercury-SP and the standard variant Mercury in C++ using the parameters specified above. We will release our implementation as open source. The implementation builds on the SecretFlow framework [34, 54] and uses its integrated implementations of BFV [51], Cheetah [26], Ferret [58], OKVS [45], and receiver-side Cuckoo hashing [31]. The sender-side simple table is built directly from the same public hash functions. Both variants use the HE interface described in Appendix C.2. For both Mercury variants, each party runs in a single thread. Their reported runtimes include protocol-internal preprocessing but exclude input generation and external key generation. The baseline measurements come from separate evaluations on the same platform.

For Yang et al. [59], we use the basic (non-prefix) L_2 implementation, which the authors report to be faster than the prefix-based variant for small δ . The reported communication ratios against this work refer to its basic non-prefix implementation. We use four planted matches and run ten independent executions after one warm-up per configuration, with fresh inputs and preprocessing for each execution. Its timer excludes input-dependent preprocessing, so these online times are not directly comparable with our protocol times. Our local three-dimensional adaptation passes the planted-match index checks but omits the third coordinate during output transfer. These incomplete-output measurements are marked ‡ in Table 2 and are excluded from best-value highlighting and performance ratios.

5.3 Experimental Setup

We run both parties on one machine with two Intel Xeon Platinum 8358 processors at 2.60 GHz and approximately 2 TiB of RAM. We emulate network conditions on the loopback interface using Linux traffic control. The nominal bandwidth and round-trip time (RTT) are 10 Gbit/s and 0.2 ms for LAN, and 100 Mbit/s and 80 ms for WAN.

5.4 Performance Comparison

We restrict our experimental comparison to prior works that provide open-source implementations [20, 2, 43, 14, 59]. For comparisons with standard FPSI baselines, Mercury provides the corresponding output functionality, while Mercury-SP is reported separately as the sender-private variant. Performance ratios refer to Mercury-SP unless stated otherwise.

Balanced Setting Table 2 reports LAN performance for equal-sized sets of 2^8 , 2^{10} , and 2^{12} points, with $d \in \{2, 3\}$ and $\delta = 32$. For Piske et al. [43], we write ρ_η^Y for the public bound on the number of receiver points that spatial hashing maps to one target block, denoted by ρ_η^R in their notation. Appendix D defines this density and explains why receiver cell sparsity gives the bound 3^d .

Table 2. Balanced-input performance for $d \in \{2, 3\}$ and $\delta = 32$ in the LAN setting. **Bold orange** and underlined blue indicate the lowest and second-lowest reported values per configuration and metric. One MiB equals 2^{20} bytes. The entries for [59] report mean online times over ten independent runs, excluding input-dependent preprocessing. A dash indicates unsupported dimension. ‡ marks locally adapted measurements that omit the third output coordinate and are excluded from ranking highlights and performance ratios.

$N_X = N_Y$	Prot.	d			
		2		3	
		Comm. (MiB)	Time (s)	Comm. (MiB)	Time (s)
2^8	GQL+25[20]	23.07	8.76	34.38	12.41
	DZL25[14]	33.16	12.09	163.91	15.65
	PST+25[43]($\rho_\eta^Y = 3^d$)	4.43	<u>1.17</u>	-	-
	YHW+26[59]	7.26	0.95	9.46 [‡]	1.00 [‡]
	Mercury-SP	3.83	1.34	7.17	1.63
	Mercury	<u>4.29</u>	1.35	<u>11.04</u>	<u>1.88</u>
2^{10}	GQL+25[20]	92.27	35.12	137.52	51.14
	DZL25[14]	132.62	46.49	655.62	63.531
	PST+25[43]($\rho_\eta^Y = 3^d$)	17.53	5.47	-	-
	YHW+26[59]	16.68	1.29	22.25 [‡]	1.33 [‡]
	Mercury-SP	7.22	<u>1.66</u>	20.01	2.80
	Mercury	<u>9.05</u>	1.76	<u>35.40</u>	<u>3.65</u>
2^{12}	GQL+25[20]	369.07	144.99	550.05	208.87
	DZL25[14]	530.49	187.21	2622.49	254.42
	PST+25[43]($\rho_\eta^Y = 3^d$)	69.81	24.86	-	-
	YHW+26[59]	54.66	2.23	76.89 [‡]	2.50 [‡]
	Mercury-SP	21.31	<u>2.93</u>	72.20	7.54
	Mercury	<u>28.60</u>	3.13	<u>133.62</u>	<u>10.98</u>

We benchmark their implementation [19] with $\rho_\eta^Y = 3^d$ in the balanced setting. These measurements use separately generated cell-sparse inputs with the same set sizes and distance thresholds as our experiments. The two protocols retain distinct input conditions.

For $d = 2$ in Table 2, Mercury-SP reduces communication by $1.2 \sim 3.3\times$ compared with [43] at $\rho_\eta^Y = 9$, and achieves $3.3 \sim 8.5\times$ runtime speedups for $N_X = N_Y \in \{2^{10}, 2^{12}\}$. At $N_X = N_Y = 2^8$, however, Mercury-SP takes 1.34 seconds, compared with 1.17 seconds for [43]. Compared with [14] across all three two-dimensional configurations, Mercury-SP achieves a $9.0 \sim 63.9\times$ runtime speedup and reduces communication by $8.7 \sim 24.9\times$. For $d = 3$, the

Table 3. LAN performance for $d = 2$, $N_X = N_Y = 2^{10}$, and $\delta \in \{32, 64, 128\}$. **Bold orange** and **underlined blue** indicate the lowest and second-lowest reported values per configuration and metric. The entries for [59] report mean online times over ten independent runs, excluding input-dependent preprocessing.

$N_X = N_Y$	Proto.	δ					
		32		64		128	
		Comm. (MiB)	Time (s)	Comm. (MiB)	Time (s)	Comm. (MiB)	Time (s)
2^{10}	[20]	92.27	35.12	181.55	66.23	359.95	126.20
	[2]	53.82	46.27	116.82	91.94	272.82	183.33
	[14]	132.62	46.492	160.564	63.32	191.653	85.388
	[43]($\rho_\eta^Y = 3^d$)	17.53	5.47	29.25	5.96	51.98	7.13
	[59]	16.68	1.29	24.90	1.33	41.26	1.42
	Mercury-SP	7.22	<u>1.66</u>	7.22	<u>1.64</u>	7.34	<u>1.67</u>
	Mercury	<u>9.05</u>	1.76	<u>9.05</u>	1.71	<u>9.16</u>	1.76

corresponding improvements over [14] are $9.6 \sim 33.7\times$ in runtime and $22.9 \sim 36.3\times$ in communication.

Against the basic non-prefix implementation of Yang et al. [59], **Mercury-SP** reduces communication by $1.9 \sim 2.6\times$ in the evaluated two-dimensional settings with $\delta = 32$. Yang et al. report shorter online times, but the different timing scopes described above preclude an end-to-end runtime comparison.

Table 3 varies $\delta \in \{32, 64, 128\}$ with $d = 2$ and $N_X = N_Y = 2^{10}$ in the LAN setting. Over this range, **Mercury-SP** takes $1.64 \sim 1.67$ seconds and transmits $7.22 \sim 7.34$ MiB, while the communication of the non-prefix baseline [59] increases from 16.68 to 41.26 MiB. At $\delta = 128$, **Mercury-SP** and the standard variant **Mercury** reduce communication by $5.6\times$ and $4.5\times$, respectively, compared with this baseline. At the same threshold, **Mercury-SP** achieves a $51.1\times$ runtime speedup and reduces communication by $26.1\times$ compared with [14].

Unbalanced Setting Tables 4 and 5 compare with [2, 43] for unbalanced inputs, with $d = 2$ and $\delta = 32$. We vary $N_Y \in \{2^8, 2^{10}, 2^{12}\}$ at fixed $N_X = 2^{18}$, and $N_X \in \{2^{16}, 2^{18}, 2^{20}\}$ at fixed $N_Y = 2^{10}$. As in the balanced setting, the Piske et al. baseline uses receiver neighborhood density $\rho_\eta^Y = 3^d = 9$. This baseline uses separately generated cell-sparse inputs with the same set sizes and distance threshold as our experiments. The $2^{18} \times 2^8$ baseline was not run, and the $2^{20} \times 2^{10}$ baseline timed out after four hours in LAN and was not run in WAN. For fixed $N_X = 2^{18}$, **Mercury-SP** achieves a $1.7 \sim 8.9\times$ LAN runtime speedup and a $5.2 \sim 9.5\times$ WAN runtime speedup over [2], while reducing communication by $9.1 \sim 14.7\times$. For $N_Y = 2^{10}$ and $N_X \in \{2^{16}, 2^{18}\}$, **Mercury-SP** achieves $77.6 \sim 162.3\times$ LAN runtime speedups and $55.7 \sim 132.4\times$ WAN runtime speedups over [43], while reducing communication by $25.6 \sim 39.3\times$.

The standard variant **Mercury** incurs additional costs for output recovery and deduplication. For $N_Y = 2^{10}$ and $N_X \in \{2^{16}, 2^{18}\}$, it achieves $66.9 \sim 138.0\times$

Table 4. Unbalanced-input performance for $N_X = 2^{18}$, $d = 2$, and $\delta = 32$. **Bold orange** and **underlined blue** indicate the lowest and second-lowest reported values per configuration and metric. Communication is measured in the LAN setting. Measurements for [43] use one run per configuration and network setting. A dash denotes an unavailable measurement.

N_X	Prot	N_Y								
		256			1024			4096		
		Comm. (MiB)	Time (s)		Comm. (MiB)	Time (s)		Comm. (MiB)	Time (s)	
			LAN	WAN		LAN	WAN		LAN	WAN
2^{18}	[2]	1310.18	30.37	121.45	1346.74	64.78	156.64	1492.99	202.97	308.89
	[43]($\rho_\eta^Y = 3^d$)	-	-	-	4396.14	3119.46	3489.99	4396.84	3116.91	3479.04
	Mercury	<u>179.82</u>	<u>21.03</u>	<u>32.10</u>	<u>202.93</u>	<u>22.60</u>	<u>34.55</u>	<u>303.91</u>	<u>27.29</u>	<u>45.27</u>
	Mercury-SP	89.06	17.41	23.51	111.91	19.23	26.36	164.47	22.92	32.51

Table 5. Unbalanced-input performance for $N_Y = 2^{10}$, $d = 2$, and $\delta = 32$. **Bold orange** and **underlined blue** indicate the lowest and second-lowest reported values per configuration and metric. Communication is measured in the LAN setting. Measurements for [43] use one run per configuration and network setting. A dash denotes an unavailable measurement. † marks WAN runs with extended timeouts and parallel chunk transmission.

N_Y	Prot	N_X								
		2^{16}			2^{18}			2^{20}		
		Comm. (MiB)	Time (s)		Comm. (MiB)	Time (s)		Comm. (MiB)	Time (s)	
			LAN	WAN		LAN	WAN		LAN	WAN
2^{10}	[2]	373.25	50.74	76.83	1346.74	64.78	156.64	5240.72	120.98	486.97
	[43]($\rho_\eta^Y = 3^d$)	1099.55	511.31	594.93	4396.14	3119.46	3489.99	-	-	-
	Mercury	<u>77.83</u>	<u>7.65</u>	<u>13.35</u>	<u>202.93</u>	<u>22.60</u>	<u>34.55</u>	<u>707.72</u>	<u>82.26</u>	<u>128.14</u> †
	Mercury-SP	42.90	6.59	10.68	111.91	19.23	26.36	346.93	67.73	91.13 †

LAN runtime speedups and $44.6 \sim 101.0\times$ WAN runtime speedups over [43], while reducing communication by $14.1 \sim 21.7\times$. At $N_X = 2^{20}$ and $N_Y = 2^{10}$, Mercury-SP takes 67.73 seconds in LAN and 91.13 seconds in WAN, with 346.93 MiB of LAN communication. The corresponding costs for the standard variant are 82.26 seconds, 128.14 seconds, and 707.72 MiB.

Acknowledgments

We used GPT 6 Astra to improve the language and clarity of this manuscript. The authors take full responsibility for its content.

References

1. Albrecht, M., Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., Halevi, S., Hoffstein, J., Laine, K., Lauter, K., Lokam, S., Micciancio, D., Moody, D., Morrison, T., Sahai, A., Vaikuntanathan, V.: Homomorphic Encryption Standard, pp. 31–62. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-77287-1_2, https://doi.org/10.1007/978-3-030-77287-1_2
2. van Baarsen, A., Pu, S.: Fuzzy private set intersection with large hyperballs. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024, Part V. LNCS, vol. 14655, pp. 340–369. Springer, Cham, Switzerland, Zurich, Switzerland (May 26–30, 2024). https://doi.org/10.1007/978-3-031-58740-5_12
3. Baldi, P., Baronio, R., De Cristofaro, E., Gasti, P., Tsudik, G.: Countering GATTACA: efficient and secure testing of fully-sequenced human genomes. In: Chen, Y., Danezis, G., Shmatikov, V. (eds.) ACM CCS 2011. pp. 691–702. ACM Press, Chicago, Illinois, USA (Oct 17–21, 2011). <https://doi.org/10.1145/2046707.2046785>
4. Berenbrink, P., Friedetzky, T., Hu, Z., Martin, R.: On weighted balls-into-bins games. *Theoretical Computer Science* **409**(3), 511–520 (2008). <https://doi.org/https://doi.org/10.1016/j.tcs.2008.09.023>, <https://www.sciencedirect.com/science/article/pii/S0304397508006713>
5. Bienstock, A., Patel, S., Seo, J.Y., Yeo, K.: Near-optimal oblivious key-value stores for efficient PSI, PSU and volume-hiding multi-maps. In: Calandrino, J.A., Troncoso, C. (eds.) USENIX Security 2023. pp. 301–318. USENIX Association, Anaheim, CA, USA (Aug 9–11, 2023), <https://www.usenix.org/conference/usenixsecurity23/presentation/bienstock>
6. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical GapSVP. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 868–886. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2012). https://doi.org/10.1007/978-3-642-32009-5_50
7. Bui, D., Garimella, G., Miao, P., Pham, P.V.L.: New framework for structure-aware PSI from distributed function secret sharing. *Cryptology ePrint Archive*, Report 2025/907 (2025), <https://eprint.iacr.org/2025/907>
8. de Castro, L., Kim, D., Kim, M., Lee, K., Min, S., Song, Y.: More efficient lattice-based OLE from circuit-private linear HE with polynomial overhead. *Cryptology ePrint Archive*, Report 2024/1534 (2024), <https://eprint.iacr.org/2024/1534>
9. Chakraborti, A., Fanti, G., Reiter, M.K.: Distance-aware private set intersection. In: Calandrino, J.A., Troncoso, C. (eds.) USENIX Security 2023. pp. 319–336. USENIX Association, Anaheim, CA, USA (Aug 9–11, 2023), <https://www.usenix.org/conference/usenixsecurity23/presentation/chakraborti-intersection>
10. Chase, M., Miao, P.: Private set intersection in the internet setting from lightweight oblivious PRF. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part III. LNCS, vol. 12172, pp. 34–63. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 17–21, 2020). https://doi.org/10.1007/978-3-030-56877-1_2
11. Chmielewski, L., Hoepman, J.H.: Fuzzy private matching (extended abstract). In: 2008 Third International Conference on Availability, Reliability and Security. pp. 327–334 (2008). <https://doi.org/10.1109/ARES.2008.170>
12. Chongchitmate, W., Lu, S., Ostrovsky, R.: Approximate PSI with near-linear communication. *Cryptology ePrint Archive*, Report 2024/682 (2024), <https://eprint.iacr.org/2024/682>

13. Cong, K., Moreno, R.C., da Gama, M.B., Dai, W., Iliashenko, I., Laine, K., Rosenberg, M.: Labeled PSI from homomorphic encryption with reduced computation and communication. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021. pp. 1135–1150. ACM Press, Virtual Event, Republic of Korea (Nov 15–19, 2021). <https://doi.org/10.1145/3460120.3484760>
14. Dang, C., Zhou, X., Liang, B.: Efficient fuzzy PSI based on prefix representation. In: Huang, C.Y., Chen, J.C., Shieh, S.P., Lie, D., Cortier, V. (eds.) ACM CCS 2025. pp. 2204–2218. ACM Press, Taipei, Taiwan (Oct 13–17, 2025). <https://doi.org/10.1145/3719027.3765203>
15. Demirel, Ö., Sbalzarini, I.F.: Balanced offline allocation of weighted balls into bins. CoRR **abs/1304.2881** (2013), <http://arxiv.org/abs/1304.2881>
16. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144 (2012), <https://eprint.iacr.org/2012/144>
17. Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword search and oblivious pseudorandom functions. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 303–324. Springer Berlin Heidelberg, Germany, Cambridge, MA, USA (Feb 10–12, 2005). https://doi.org/10.1007/978-3-540-30576-7_17
18. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 1–19. Springer Berlin Heidelberg, Germany, Interlaken, Switzerland (May 2–6, 2004). https://doi.org/10.1007/978-3-540-24676-3_1
19. daot-fpsi. <https://github.com/asu-crypto/daOT-fuzzyPSI> (Jun 2025), <https://github.com/asu-crypto/daOT-fuzzyPSI>
20. Gao, Y., Qi, L., Liu, X., Luo, Y., Wang, L.: Efficient fuzzy private set intersection from fuzzy mapping. In: Chung, K.M., Sasaki, Y. (eds.) ASIACRYPT 2024, Part VI. LNCS, vol. 15489, pp. 36–68. Springer, Singapore, Singapore, Kolkata, India (Dec 9–13, 2024). https://doi.org/10.1007/978-981-96-0938-3_2
21. Garimella, G., Goff, B., Miao, P.: Computation efficient structure-aware PSI from incremental function secret sharing. In: Reyzin, L., Stebila, D. (eds.) CRYPTO 2024, Part VIII. LNCS, vol. 14927, pp. 309–345. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 18–22, 2024). https://doi.org/10.1007/978-3-031-68397-8_10
22. Garimella, G., Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: Oblivious key-value stores and amplification for private set intersection. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part II. LNCS, vol. 12826, pp. 395–425. Springer, Cham, Switzerland, Virtual Event (Aug 16–20, 2021). https://doi.org/10.1007/978-3-030-84245-1_14
23. Garimella, G., Rosulek, M., Singh, J.: Structure-aware private set intersection, with applications to fuzzy matching. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part I. LNCS, vol. 13507, pp. 323–352. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 15–18, 2022). https://doi.org/10.1007/978-3-031-15802-5_12
24. Garimella, G., Rosulek, M., Singh, J.: Malicious secure, structure-aware private set intersection. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part I. LNCS, vol. 14081, pp. 577–610. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 20–24, 2023). https://doi.org/10.1007/978-3-031-38557-5_19
25. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) 41st ACM STOC. pp. 169–178. ACM Press, Bethesda, MD, USA (May 31 – Jun 2, 2009). <https://doi.org/10.1145/1536414.1536440>

26. Huang, Z., Jie Lu, W., Hong, C., Ding, J.: Cheetah: Lean and fast secure two-party deep neural network inference. In: Butler, K.R.B., Thomas, K. (eds.) *USENIX Security 2022*. pp. 809–826. USENIX Association, Boston, MA, USA (Aug 10–12, 2022), <https://www.usenix.org/conference/usenixsecurity22/presentation/huang-zhicong>
27. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) *CRYPTO 2003*. LNCS, vol. 2729, pp. 145–161. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2003). https://doi.org/10.1007/978-3-540-45146-4_9
28. Kales, D., Rechberger, C., Schneider, T., Senker, M., Weinert, C.: Mobile private contact discovery at scale. In: Heninger, N., Traynor, P. (eds.) *USENIX Security 2019*. pp. 1447–1464. USENIX Association, Santa Clara, CA, USA (Aug 14–16, 2019), <https://www.usenix.org/conference/usenixsecurity19/presentation/kales>
29. Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious PRF with applications to private set intersection. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) *ACM CCS 2016*. pp. 818–829. ACM Press, Vienna, Austria (Oct 24–28, 2016). <https://doi.org/10.1145/2976749.2978381>
30. Kolesnikov, V., Matania, N., Pinkas, B., Rosulek, M., Trieu, N.: Practical multi-party private set intersection from symmetric-key techniques. In: Thuraishingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) *ACM CCS 2017*. pp. 1257–1272. ACM Press, Dallas, TX, USA (Oct 31 – Nov 2, 2017). <https://doi.org/10.1145/3133956.3134065>
31. Microsoft Kuku (release 2.1.0). <https://github.com/microsoft/Kuku> (Feb 2021), microsoft Research, Redmond, WA.
32. Kulshrestha, A., Mayer, J.R.: Identifying harmful media in end-to-end encrypted communication: Efficient private membership computation. In: Bailey, M., Greenstadt, R. (eds.) *USENIX Security 2021*. pp. 893–910. USENIX Association (Aug 11–13, 2021), <https://www.usenix.org/conference/usenixsecurity21/presentation/kulshrestha>
33. Lindell, Y.: How to simulate it - A tutorial on the simulation proof technique. *Cryptology ePrint Archive*, Report 2016/046 (2016), <https://eprint.iacr.org/2016/046>
34. Ma, J., Zheng, Y., Feng, J., Zhao, D., Wu, H., Fang, W., Tan, J., Yu, C., Zhang, B., Wang, L.: SecretFlow-SPU: A performant and User-Friendly framework for Privacy-Preserving machine learning. In: *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. pp. 17–33. USENIX Association, Boston, MA (Jul 2023), <https://www.usenix.org/conference/atc23/presentation/ma>
35. Meadows, C.: A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In: *Proceedings of the 1986 IEEE Symposium on Security and Privacy*, Oakland, California, USA, April 7–9, 1986. pp. 134–137. IEEE Computer Society (1986). <https://doi.org/10.1109/SP.1986.10022>, <https://doi.org/10.1109/SP.1986.10022>
36. Miao, P., Patel, S., Raykova, M., Seth, K., Yung, M.: Two-sided malicious security for private intersection-sum with cardinality. In: Micciancio, D., Ristenpart, T. (eds.) *CRYPTO 2020, Part III*. LNCS, vol. 12172, pp. 3–33. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 17–21, 2020). https://doi.org/10.1007/978-3-030-56877-1_1
37. Pagh, R., Rodler, F.F.: Cuckoo hashing. In: auf der Heide, F.M. (ed.) *Algorithms — ESA 2001*. pp. 121–133. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)

38. Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: SpOT-light: Lightweight private set intersection from sparse OT extension. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 401–431. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 18–22, 2019). https://doi.org/10.1007/978-3-030-26954-8_13
39. Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: PSI from PaXoS: Fast, malicious private set intersection. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part II. LNCS, vol. 12106, pp. 739–767. Springer, Cham, Switzerland, Zagreb, Croatia (May 10–14, 2020). https://doi.org/10.1007/978-3-030-45724-2_25
40. Pinkas, B., Schneider, T., Zohner, M.: Faster private set intersection based on OT extension. In: Fu, K., Jung, J. (eds.) USENIX Security 2014. pp. 797–812. USENIX Association, San Diego, CA, USA (Aug 20–22, 2014), <https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/pinkas>
41. Pinkas, B., Schneider, T., Zohner, M.: Scalable private set intersection based on ot extension. ACM Trans. Priv. Secur. **21**(2) (Jan 2018). <https://doi.org/10.1145/3154794>, <https://doi.org/10.1145/3154794>
42. Piske, L., van de Graaf, J., Nascimento, A.C.A., Trieu, N.: Shared OT and its applications. IACR Communications in Cryptology **2**(2) (2025). <https://doi.org/10.62056/a0wa3z10k>
43. Piske, L., Singh, J., Trieu, N., Kolesnikov, V., Zikas, V.: Distance-aware OT with application to fuzzy PSI. In: Huang, C.Y., Chen, J.C., Shieh, S.P., Lie, D., Cortier, V. (eds.) ACM CCS 2025. pp. 4679–4691. ACM Press, Taipei, Taiwan (Oct 13–17, 2025). <https://doi.org/10.1145/3719027.3744857>
44. Rabin, M.O.: How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Report 2005/187 (2005), <https://eprint.iacr.org/2005/187>
45. Raghuraman, S., Rindal, P.: Blazing fast PSI from improved OKVS and sub-field VOLE. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022. pp. 2505–2517. ACM Press, Los Angeles, CA, USA (Nov 7–11, 2022). <https://doi.org/10.1145/3548606.3560658>
46. Richardson, D., Rosulek, M., Xu, J.: Fuzzy PSI via oblivious protocol routing. Cryptology ePrint Archive, Report 2024/1642 (2024), <https://eprint.iacr.org/2024/1642>
47. Rindal, P., Schoppmann, P.: VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part II. LNCS, vol. 12697, pp. 901–930. Springer, Cham, Switzerland, Zagreb, Croatia (Oct 17–21, 2021). https://doi.org/10.1007/978-3-030-77886-6_31
48. Rivest, R.L., Adleman, L., Dertouzos, M.L., et al.: On data banks and privacy homomorphisms. Foundations of secure computation **4**(11), 169–180 (1978)
49. Rosulek, M., Trieu, N.: Compact and malicious private set intersection for small sets. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021. pp. 1166–1181. ACM Press, Virtual Event, Republic of Korea (Nov 15–19, 2021). <https://doi.org/10.1145/3460120.3484778>
50. Schroff, F., Kalenichenko, D., Philbin, J.: Facenet: A unified embedding for face recognition and clustering. CoRR **abs/1503.03832** (2015), <http://arxiv.org/abs/1503.03832>
51. Microsoft SEAL (release 4.0). <https://github.com/Microsoft/SEAL> (Mar 2022), microsoft Research, Redmond, WA.
52. Smart, N.P., Vercauteren, F.: Fully homomorphic SIMD operations. DCC **71**(1), 57–81 (2014). <https://doi.org/10.1007/s10623-012-9720-4>

53. Son, H., Paik, S., Kim, Y., Kim, S., Chung, H., Seo, J.H.: Doubly efficient fuzzy private set intersection for high-dimensional data with cosine similarity. *IEEE Access* **13**, 217108–217125 (2025). <https://doi.org/10.1109/ACCESS.2025.3648455>
54. Secretflow-spu. <https://github.com/secretflow/spu> (Jul 2023)
55. Uzun, E., Chung, S.P., Kolesnikov, V., Boldyreva, A., Lee, W.: Fuzzy labeled private set intersection with applications to private real-time biometric search. In: Bailey, M., Greenstadt, R. (eds.) *USENIX Security 2021*. pp. 911–928. USENIX Association (Aug 11–13, 2021), <https://www.usenix.org/conference/usenixsecurity21/presentation/uzun>
56. van Baarsen, A., Pu, S.: Fuzzy private set intersection from VOLE. In: Hanaoka, G., Yang, B.Y. (eds.) *ASIACRYPT 2025, Part V*. LNCS, vol. 16249, pp. 327–360. Springer, Singapore, Singapore, Melbourne, VIC, Australia (Dec 8–12, 2025). https://doi.org/10.1007/978-981-95-5116-3_11
57. Wang, C., Zheng, W., Li, J., Zhou, J., Lu, J.: Deep factorized metric learning. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. pp. 7672–7682 (June 2023)
58. Yang, K., Weng, C., Lan, X., Zhang, J., Wang, X.: Ferret: Fast extension for correlated OT with small communication. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) *ACM CCS 2020*. pp. 1607–1626. ACM Press, Virtual Event, USA (Nov 9–13, 2020). <https://doi.org/10.1145/3372297.3417276>
59. Yang, X., Hao, M., Weng, C., Deng, R.H., Wen, Y., Zhang, T.: Efficient Fuzzy Private Set Intersection from Secret-shared OPRF . In: *2026 IEEE Symposium on Security and Privacy (SP)*. pp. 2008–2027. IEEE Computer Society, Los Alamitos, CA, USA (May 2026). <https://doi.org/10.1109/SP63933.2026.00172>, <https://doi.ieeecomputersociety.org/10.1109/SP63933.2026.00172>
60. Ye, Q., Steinfeld, R., Pieprzyk, J., Wang, H.: Efficient fuzzy matching and intersection on private datasets. In: Lee, D., Hong, S. (eds.) *ICISC 09*. LNCS, vol. 5984, pp. 211–228. Springer Berlin Heidelberg, Germany, Seoul, Korea (Dec 2–4, 2010). https://doi.org/10.1007/978-3-642-14423-3_15
61. Yeo, K.: Cuckoo hashing in cryptography: Optimal parameters, robustness and applications. In: Handschuh, H., Lysyanskaya, A. (eds.) *CRYPTO 2023, Part IV*. LNCS, vol. 14084, pp. 197–230. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 20–24, 2023). https://doi.org/10.1007/978-3-031-38551-3_7
62. Zhang, C., Chen, Y., Cao, Y., Bai, Y., Li, S., Lin, J., Wang, A., Wang, X.: Fast fuzzy PSI from symmetric-key techniques. *Cryptology ePrint Archive*, Report 2025/885 (2025), <https://eprint.iacr.org/2025/885>

A Extensions of Our FPSI Protocol

In this section, we discuss several extensions of Mercury.

A.1 Other Distance Metrics

The construction in the main text is instantiated for the L_2 distance. Its spatial-hashing layer extends to another metric whenever the corresponding activation rule preserves every pair within distance δ . The remaining change is confined to batched distance evaluation and the secure masked-range test.

For the L_∞ distance, a pair is neighboring exactly when $|x[\ell] - y[\ell]| \leq \delta$ for every coordinate $\ell \in [d]$. An L_∞ instantiation can therefore replace the scalar $\mathcal{F}_{\text{BlockDistShare}}$ and $\mathcal{F}_{\text{SMRT}}$ interfaces with a block-selective vector-sharing interface followed by the conjunction of the coordinatewise range predicates. Each variant retains its own block-matching interface and output procedure.

A.2 Functionality Extensions

Mercury supports several extensions of FPSI. Circuit FPSI replaces the output operation in Step 7 with the desired circuit over the validity-filtered predicates. Labeled FPSI uses the Non-SP construction variant, associating each sender record with its label and supplying point-label pairs to the output OT step of Fig. 15. FPSI-SP cardinality first computes one predicate $Ins[i]$ for each receiver bin as in Step 7. The parties then convert these Boolean shares to additive shares and sum them, which counts each reported receiver point once even when it has several neighboring sender points.

B Realization of the Secure Masked-Range Test

The secure masked-range test in Section 4.3 receives one additive squared-distance share from each party and returns Boolean shares of the cyclic range predicate. We instantiate this functionality with the secure comparison and equality primitives from the preliminaries. These primitives are implemented with the Cheetah 2PC comparison circuit [26]. The full FPSI protocol invokes the test with range bound $\tau = \delta^2$ because the shared value is a squared L_2 distance. The implementation uses the equivalent wrap test $x \geq p - \tau$, including the equality case, to avoid overflow when evaluating $x + \tau$.

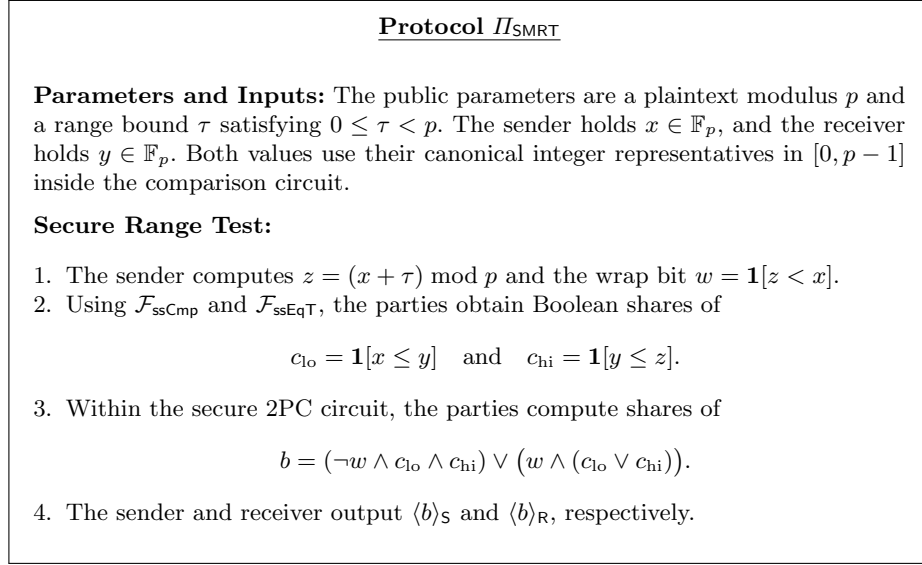


Fig. 14. A 2PC realization of the secure masked-range test from secret-shared comparison and equality.

Theorem 5. *Protocol Π_{SMRT} in Fig. 14 outputs Boolean shares of $\mathbf{1}[y \in \mathcal{I}_p(x, \tau)]$, where $\mathcal{I}_p(x, \tau)$ is defined in Fig. 12.*

Proof. Suppose first that $x + \tau < p$. Then $z = x + \tau$ and $w = 0$, so the output formula reduces to $b = c_{\text{lo}} \wedge c_{\text{hi}}$. Hence, $b = 1$ exactly when $x \leq y \leq x + \tau$, which is equivalent to $y \in \mathcal{I}_p(x, \tau)$.

Suppose instead that $x + \tau \geq p$. Then $z = x + \tau - p$ and $w = 1$, so the output formula reduces to $b = c_{\text{lo}} \vee c_{\text{hi}}$. Hence, $b = 1$ exactly when $y \in [x, p-1]$ or $y \in [0, z]$, which is the wrapped interval $\mathcal{I}_p(x, \tau)$.

Theorem 6. *Protocol Π_{SMRT} securely realizes $\mathcal{F}_{\text{SMRT}}$ against semi-honest adversaries, provided that $\mathcal{F}_{\text{ssCmp}}$, $\mathcal{F}_{\text{ssEqT}}$, and the remaining circuit logic in Fig. 14 are securely realized by 2PC.*

Proof. All intermediate comparisons and Boolean values remain secret shared. The only prescribed outputs are Boolean shares of b . The standard sequential-composition theorem therefore gives a simulator by composing the simulators of the comparison and equality subprotocols with that of the remaining 2PC circuit. Theorem 5 shows that the reconstructed bit equals the output of $\mathcal{F}_{\text{SMRT}}$.

C Security Proofs

This appendix provides the security proof details omitted from the main text.

C.1 Security of Block-Matching Hints

The sender assigns its blocks uniformly to distinct auxiliary rows and fills all unused entries with independent uniform values in the corresponding domains. The hint identifies a protected row carrying the segment-start index, while the masked validity vector is stored either in the hint or in a separate protected column. The start index minus one is protected by addition modulo C_{pt} with a row-bound PRF pad, while validity vectors are protected by XOR. All auxiliary values are encoded in their full public domains, so decoding with a nonmatching key still produces legal indices and does not expose an additional failure flag.

Proof (Theorem 2). For a matching sender block j in bin i , OPRF consistency and OKVS correctness allow the receiver to recover the segment-start index $ST[i][j]$, the seed $\sigma[i]$, and the masked validity vector $u[i] \oplus f[i][j]$ by cancelling the corresponding PRF pads. The secure 2PC therefore returns Boolean shares of $eq[i] = 1$ and $val[i][k] = f[i][j][k]$, as prescribed by $\mathcal{F}_{BMH}$.

Each hybrid retains all parts of its predecessor except for the stated replacements and their dependent computations.

We first consider a corrupted sender and define the following hybrids.

- H_0^S . This is the sender’s view in the real protocol, including its local randomness and the OPRF and 2PC transcripts.
- H_1^S . Replace the OPRF transcript in H_0^S with its simulated transcript for the public N_Y -query batch, keeping the sender’s input tables, keys, and local masks unchanged.
- H_2^S . Replace the seed-equality and validity-computation transcripts in H_1^S with their 2PC simulations on the sender’s inputs and prescribed Boolean output shares. Use the same local seeds and masks as in the preceding hybrid.

The ideal-world sender simulator receives BT_S , ST , and the sender’s output shares from $\mathcal{F}_{BMH}$. It generates the sender’s state honestly and invokes the OPRF and 2PC simulators as in H_2^S , without using the receiver’s block table.

The sender hybrids are computationally indistinguishable for the following reasons.

- H_0^S and H_1^S . OPRF query privacy gives $H_0^S \approx_c H_1^S$, since the simulated OPRF transcript requires only the sender’s key and the public batch size.
- H_1^S and H_2^S . Secure 2PC gives $H_1^S \approx_c H_2^S$, since its simulators receive the sender’s own inputs and prescribed output shares.

For a corrupted receiver, we use the following hybrids.

- H_0^R . This is the receiver’s real view, including its OPRF outputs, the complete OKVS encodings and auxiliary rows, its decoded values, and the 2PC transcripts and output shares.
- H_1^R . Replace the OPRF and 2PC transcripts in H_0^R by their simulations on the receiver’s actual inputs and outputs. Keep all receiver-known block keys and their domain-separated evaluations unchanged.

- H_2^R . Replace PRF evaluations under receiver-unknown block keys with uniform values in their specified domains. Reuse the same sampled value whenever the key, domain, and input repeat, and retain every evaluation under a receiver-known key. Recompute the affected encodings, decoded values, and simulated transcripts consistently with these replacements.
- H_3^R . Replace the generation of the complete OKVS encodings and auxiliary rows by the following procedure, using only the receiver’s block keys and prescribed outputs from $\mathcal{F}_{\text{BMH}}$.
 1. For each nonempty receiver bin, take the segment-start index supplied by the functionality and sample a uniform auxiliary-row index, a uniform seed, and a uniform masked validity word.
 2. For each such bin, protect the prescribed start index in the chosen row using the receiver’s block key. Sample the entire OKVS uniformly subject to the decoding equation that yields the chosen row and the sampled payload under that key. If validity occupies a separate auxiliary column, constrain the chosen row to recover the sampled validity word under its corresponding pad. Sample all remaining auxiliary entries independently in their specified domains.
 3. For an empty receiver bin, sample the encoding and auxiliary rows without a decoding constraint, make no OPRF query, and retain the prescribed local decoding defaults.
 4. In every bin, run the 2PC simulators on the resulting receiver inputs and the Boolean output shares supplied by the functionality. All additional lookups use these same sampled structures.

This defines the ideal-world receiver simulator given BT_R , the prescribed segment-start indices and Boolean shares, and the public parameters. The receiver’s OPRF outputs are generated according to their prescribed distribution and reused throughout the simulated view. Neither the sender’s block table nor the reconstructed match indicators and validity vectors are needed to generate H_3^R .

We justify the receiver transitions as follows.

- H_0^R and H_1^R . The security of the OPRF and 2PC gives $H_0^R \approx_c H_1^R$, with the keys, decoded values, and output shares preserved across the simulations.
- H_1^R and H_2^R . Multi-user PRF security across all derivation domains gives $H_1^R \approx_c H_2^R$, because only evaluations under receiver-unknown keys are replaced.
- H_2^R and H_3^R . We compare the two sampling procedures one bin at a time.
 1. The hash-function index binds each receiver-known block key to its bin, so at most one such key corresponds to a real sender block in that bin. In H_2^R , independent uniform pads protect all other blocks, even though their unprotected payloads share the same seed and validity mask. For a matching block j , the seed and assigned auxiliary row are uniform, and $u[i] \oplus f[i][j]$ is uniform independently of the hidden segment length. These are precisely the values sampled in the first step of H_3^R .
 2. Write the linear OKVS equations as $A_{\text{kv}}P = V_{\text{kv}}$, where P is the complete encoding and V_{kv} contains the protected values. We require full row

rank and use the private encoding randomness ρ_{enc} to sample the free variables uniformly, independently of V_{kv} and the public decoding seed θ . The uniform protected values and free variables then make P uniform. After fixing the receiver’s decoded payload, P is uniform over the affine subspace satisfying that decoding equation, exactly as sampled in H_3^R .

3. For a matching block, H_3^R protects the prescribed start index in the uniformly selected auxiliary row using the same known-key pad as H_2^R . For a nonmatching block, the required nonzero decoding row yields a uniform payload, and the independent uniform auxiliary entries yield a uniform legal start index. Sampling this index first and then setting its protected entry preserves the distribution because addition of the pad modulo C_{pt} is a bijection. A separate validity column satisfies the same argument, using XOR and the uniform decoded validity word. All remaining auxiliary entries are uniform in both hybrids. Empty receiver bins use unconstrained encodings and auxiliary rows with the same fixed decoding defaults.
4. Every additional lookup or scan under a receiver-known key uses the same sampled encoding and auxiliary rows. Their correlations with the selected values are therefore preserved, rather than simulated as independent answers. The 2PC simulators consequently receive equally distributed receiver inputs and prescribed Boolean output shares in both hybrids.

The two hybrids therefore induce the same distribution of the receiver’s view, up to the sampling errors and failure probabilities assumed negligible in Appendix C.3.

Combining the transitions relates each party’s real view to its ideal-world simulation and proves the theorem.

The same hint-simulation argument applies to standard FPSI, with the segment-start index replaced by the candidate-set rank in the public domain specified in Appendix E.1.

C.2 Security of Block-Selective Distance Sharing

We require the randomized linear evaluation based on de Castro et al. [8] to provide circuit privacy for the complete response in the presence of the receiver’s auxiliary information.

Proof (Theorem 3). We first consider a corrupted sender and define the following hybrids.

- H_0^S . This is the sender’s real view, including its input tables, OPRF-derived block keys $K_S[i][j]$, local masks, and the encrypted receiver query.
- H_1^S . Replace the OPRF transcript with its simulation, keeping the sender’s block keys and masks unchanged.
- H_2^S . Replace the receiver query in H_1^S by encryptions of zero with the same BFV key distribution and public query format. Compute the sender’s local response from this query using the same input tables, block keys, and masks.

The ideal-world sender simulator uses the shares supplied by $\mathcal{F}_{\text{BlockDistShare}}$ as $\text{mask}_{in}[i][k]$ and samples the remaining local masks honestly. Given its input tables and block keys, it generates the simulated OPRF transcript and zero query as in H_2^S , without using the receiver's input.

The sender hybrids are computationally indistinguishable for the following reasons.

- H_0^S and H_1^S . OPRF query privacy gives $H_0^S \approx_c H_1^S$.
- H_1^S and H_2^S . BFV input privacy gives $H_1^S \approx_c H_2^S$. The ideal sender shares are independent uniform field elements, so using them as $\text{mask}_{in}[i][k]$ preserves the distribution of the sender's local masks and response computation.

For a corrupted receiver, we use the following hybrids.

- H_0^R . This is the receiver's real view, including its query, BFV key pair (pk, sk) , encryption randomness, OPRF-derived block keys $K_R[i]$, complete ciphertext response, and recovered distance shares.
- H_1^R . Simulate the OPRF transcript consistently with the receiver's block keys, and replace the response ciphertexts by the circuit-privacy simulation of their actual masked plaintexts. Keep the receiver query, BFV key pair, encryption randomness, and all plaintext values unchanged.
- H_2^R . Replace distance pads under receiver-unknown block keys with uniform field elements, using one sampled value for each distinct block key and pad input. Preserve repeated evaluations and all pads under receiver-known block keys. Recompute the masked plaintext response, its simulated ciphertexts, and the selected receiver shares consistently with these replacements.
- H_3^R . Replace the generation of the complete masked response by a sampler using the receiver's input, block keys, and ideal output shares. Write $V[i][a]$ for the decrypted point-table slot after adding the receiver's squared norm. For each selected address $\alpha_{i,k}$, use the share supplied by $\mathcal{F}_{\text{BlockDistShare}}$ to set

$$V[i][\alpha_{i,k}] = \llbracket \text{Dist}[i][k] \rrbracket_R + D_{K_R[i]}(i, k).$$

Sample every other logical slot uniformly. For empty receiver bins, use the zero local pad and set the selected plaintext values directly to the ideal receiver shares. Subtract the receiver squared norm and include independently protected padding for BFV slots outside the point table to obtain the full plaintext response. Invoke the circuit-privacy simulator from H_1^R on this response and the receiver's query, BFV key pair, and encryption randomness. Use the same block keys and known-key pad evaluations as in the hint simulation.

The ideal-world receiver simulator generates its query honestly from its own input and performs the sampling in H_3^R using its prescribed distance shares and block keys. It needs neither the sender's point table nor the private evaluation coefficients, and it produces the entire ciphertext response rather than only the selected window.

We justify the receiver transitions as follows.

- $\mathbf{H}_0^R$ and $\mathbf{H}_1^R$. OPRF security and the circuit-privacy premise give $\mathbf{H}_0^R \approx_c \mathbf{H}_1^R$. This replacement covers every returned ciphertext, including evaluations of reused queries and the additional padding used for BFV batching, while preserving the receiver’s BFV key pair, block keys, and the randomness used to encrypt the query.
- $\mathbf{H}_1^R$ and $\mathbf{H}_2^R$. Multi-user PRF security gives $\mathbf{H}_1^R \approx_c \mathbf{H}_2^R$, since only pads under receiver-unknown block keys are replaced.
- $\mathbf{H}_2^R$ and $\mathbf{H}_3^R$. We verify the response distribution in three steps.
 1. At a valid selected position, matching block keys and aligned indices give

$$V[i][\alpha_{i,k}] = \text{Dist}(\tilde{x}_{i,k}, \tilde{y}_i) + \text{mask}_{in}[i][k] + D_{K_R[i]}(i, k).$$

The masks $\text{mask}_{in}[i][k]$ are independent across valid positions, so subtracting the receiver’s pad yields shares with the distribution prescribed by $\mathcal{F}_{\text{BlockDistShare}}$.

2. All other real slots use receiver-unknown block keys. In $\mathbf{H}_2^R$, each such slot has an independent uniform pad, even when $\text{mask}_{in}[i][k]$ is reused across blocks. Dummy slots also have independent uniform masks. These slots are therefore jointly uniform and independent of the valid selected values, and the selected invalid shares have the distribution prescribed by $\mathcal{F}_{\text{BlockDistShare}}$.
3. The selected addresses are distinct because $C_{\text{act}} \leq C_{\text{pt}}$. Fixing their values using the ideal receiver shares and sampling the unselected values uniformly therefore reproduces the complete plaintext distribution. Empty bins satisfy the same argument with zero local pads. Applying the same circuit-privacy simulator to this plaintext response preserves the receiver’s complete ciphertext view and its relation to the block keys.

Thus $\mathbf{H}_2^R$ and $\mathbf{H}_3^R$ induce the same distribution of the receiver’s view, up to the stated sampling errors and exceptional events.

Combining the transitions gives the two ideal-world simulations under the stated HE premise, assuming negligible sampling errors and failure probabilities.

C.3 Security of the Full FPSI-SP Protocol

Proof (Theorem 4). Let the ideal receiver output be

$$Z = \{y \in Y \mid \exists x \in X : \text{dist}(x, y) \leq \delta\}.$$

We prove security for **Mercury-SP** directly and treat standard FPSI independently in Appendix E.4. In particular, the proof for **Mercury-SP** does not open individual candidate predicates or supply sender points. Each hybrid retains the preceding view except for the stated replacements and their dependent computations.

We first consider a corrupted sender and define the following hybrids.

- $\mathbf{H}_0^S$. This is the sender’s view in the real protocol, including its local tables, masks, received query, and primitive transcripts.

- H_1^S . Replace the OPRF and 2PC transcripts by their simulations on the sender’s actual inputs and prescribed output shares. This includes equality, validity, range testing, and the final disjunction, with the same local masks and shares used across all calls.
- H_2^S . Replace the encrypted receiver coordinates in H_1^S by encryptions of zero with the same BFV key distribution and public query format. Recompute the sender’s encrypted response from this query and its unchanged local tables and masks. All subsequent simulated computations retain the same masks and shares.

The ideal-world sender simulator constructs the active blocks, padded point tables, independent cyclic rotations, and local masks honestly from X and the public configuration. It samples the sender’s Boolean output shares from their prescribed uniform distributions and uses its distance-protection masks as its distance shares. It then invokes the primitive simulators and generates the zero query as in H_2^S . The sender receives no reconstructed bin result, so the simulator requires no ideal output or receiver input.

The sender hybrids are computationally indistinguishable for the following reasons.

- H_0^S and H_1^S . OPRF query privacy and secure 2PC give $H_0^S \approx_c H_1^S$.
- H_1^S and H_2^S . BFV input privacy gives $H_1^S \approx_c H_2^S$, with the sender’s response computed consistently from the replaced query. The remaining state is generated honestly from X , and the uniform Boolean shares and retained distance masks have the same distribution as in the real execution.

For a corrupted receiver, we use the following hybrids.

- H_0^R . This is the receiver’s real view, including its local tables, query, BFV key pair, encryption randomness, block keys, complete hint and distance responses, and opened bin results.
- H_1^R . Replace the OPRF and 2PC transcripts by their simulations using the receiver’s actual inputs, shares, and opened results. Then replace the complete HE response by the circuit-privacy simulation of its actual masked plaintexts. Keep all block keys, decoded values, distance shares, and opened bits unchanged.
- H_2^R . Replace the hint and distance pads under receiver-unknown keys with independent uniform values for distinct key-domain-input combinations. Repeated evaluations use the same sampled value, and all receiver-known keys and their evaluations remain unchanged. Recompute the affected encodings, masked responses, receiver inputs, and simulated transcripts consistently with these replacements.
- H_3^R . Replace the generation of the alignment information, complete hint tables, masked response, and receiver shares by the following sampler using Y, Z . Retain the receiver’s locally generated tables, query, BFV key pair, and encryption randomness.
 1. For each nonempty receiver bin, sample an independent uniform segment-start index in $[C_{pt}]$. Empty receiver bins use the fixed local default.

2. Generate all hint encodings and auxiliary rows using the sampler in Appendix C.1, constrained by the sampled starts and actual receiver-known keys. Sample the values $V[i][a]$ from Appendix C.2 uniformly and derive the selected receiver shares using the same known-key distance pads. Subtract the receiver squared norm and include independently protected padding for BFV slots outside the point table to obtain the complete plaintext response. Apply the HE simulator from H_1^R to this response.
3. Sample the receiver-side Boolean shares according to the ideal component interfaces. For the final opening, set

$$Ins[i] = \begin{cases} 1, & \text{if bin } i \text{ contains a receiver point in } Z, \\ 0, & \text{otherwise.} \end{cases}$$

Invoke the 2PC simulators from H_1^R using the sampled receiver inputs and shares and the prescribed opened results. Return the receiver point from each bin with $Ins[i] = 1$, without releasing an individual candidate predicate.

The ideal-world receiver simulator generates its state honestly from Y , samples its OPRF outputs as in the hint simulation, and performs the sampling in H_3^R using Z . It retains one consistent set of block keys, hint values, distance shares, and 2PC inputs throughout. This procedure uses no sender input, hidden match indicator, segment length, or individual distance.

We justify the receiver transitions as follows.

- **H_0^R and H_1^R .** OPRF security, secure 2PC, and circuit privacy with auxiliary input give $H_0^R \approx_c H_1^R$.
- **H_1^R and H_2^R .** Multi-user PRF security gives $H_1^R \approx_c H_2^R$, with all receiver-known keys and repeated evaluations preserved.
- **H_2^R and H_3^R .** We verify the receiver's view in three steps.
 1. An independent cyclic rotation makes the segment-start index of a matching block uniform, regardless of its length or original location. A nonmatching real block also yields a uniform start, while an empty bin uses the prescribed fixed default. The starts sampled in H_3^R therefore have the required distribution.
 2. Appendix C.1 establishes the distribution of the complete hint encodings and auxiliary rows under the receiver-known block keys. Appendix C.2 shows that the complete vector of masked plaintext values is uniformly distributed, despite reuse of the per-bin distance-protection masks across blocks. The hint and distance pads use separate PRF domains, and both simulations preserve the same block keys and known-key evaluations. The receiver's view of the hints, masked response, and derived distance shares therefore has the same distribution in both hybrids.
 3. The receiver-side Boolean shares follow the prescribed component distributions, and the main-text correctness argument determines every opened $Ins[i]$ from Y, Z . Invalid candidates cannot affect the disjunction because their validity bits are zero. Applying the same HE and 2PC simulators to the sampled values preserves the complete transcript and its final opening.

Thus H_2^R and H_3^R have the same distribution up to the stated sampling errors and exceptional events.

The two hybrid sequences end in the ideal-world simulations described above.

Assuming that the sampling errors and failure probabilities discussed above are negligible, the preceding hybrids show that each party's real and simulated views are computationally indistinguishable when table construction succeeds, under the stated cryptographic assumptions. This proves Theorem 4.

D Input Assumptions and Candidate Multiplicity

Comparison criterion. This appendix uses X for the sender input and Y for the receiver input, consistently with the main protocol. Most subquadratic FPSI constructions first apply a coarse filter that determines which cross-party point pairs may be neighboring. An input condition makes this filtering step efficient by limiting how many points can share a geometric region, an identifier, or a routed protocol instance. To compare these conditions independently of the cryptographic primitive, let $\mathcal{F}$ denote a coarse filter and define its logical candidate set by

$$\text{Cand}_{\mathcal{F}}(X, Y) := \{(x, y) \in X \times Y : \\ \mathcal{F} \text{ assigns } x \text{ and } y \\ \text{to a common test instance}\}.$$

Completeness requires every pair (x, y) satisfying $\text{dist}(x, y) \leq \delta$ to belong to $\text{Cand}_{\mathcal{F}}(X, Y)$. We write $C_{\mathcal{F}} := |\text{Cand}_{\mathcal{F}}(X, Y)|$ for the resulting candidate multiplicity. Some protocols process identifiers or padded table positions in batch instead of materializing these pairs, but $C_{\mathcal{F}}$ still measures the number of logical pairwise tests induced by the filter. The overall complexity of an FPSI construction is largely governed by the number of point pairs that survive this preliminary filter and the cost of securely processing each pair. We therefore use candidate multiplicity $C_{\mathcal{F}}$ as a central metric for comparing the filtering effect of these input conditions and report the resulting asymptotic costs in Table 1.

No assumption. As a reference, consider a baseline that imposes no density or separation condition on either input and securely tests every cross-party point pair. Its candidate set and candidate multiplicity are

$$\text{Cand}_{\text{all}}(X, Y) = X \times Y, \quad C_{\text{all}} = N_X N_Y.$$

If one secure pairwise test costs t , the pairwise-testing component costs $N_X N_Y t$, with communication and computation accounted for separately. Without an input restriction, a geometric filter need not reduce this candidate set: whenever the domain permits both inputs to lie in a region of diameter at most δ , every cross-party pair is neighboring and must survive a complete filter. This is a baseline for explicit pairwise testing, not a quadratic lower bound for arbitrary FPSI protocols, which may aggregate tests or avoid enumerating individual pairs.

Cell- and density-based conditions. Recall from Definition 4 that ρ_c^Z is the maximum number of points of Z whose home block is the same 2δ -side grid block. Let $\text{neigh}_\delta(z)$ be the at most 2^d blocks selected by the coordinatewise activation rule in Section 4.1, applied to z . Using the neighborhood-density notation of Piske et al. [43], we count how many points this spatial-hashing rule maps to one target block:

$$\rho_\eta^Z := \max_{\mathcal{B}} |\{z \in Z \mid \mathcal{B} \in \text{neigh}_\delta(z)\}|.$$

Thus, ρ_c^Z measures home-block occupancy, whereas ρ_η^Z measures the maximum occupancy after spatial hashing. When used as a public capacity parameter, ρ_η^Z may be any upper bound on this maximum.

Two-sided cell sparsity (cell-sparse). The condition used by **Mercury** is $\rho_c^X, \rho_c^Y \leq 1$. It does not impose a minimum separation on points whose home blocks differ. Fix a receiver home block $\mathcal{B}$. Every sender point that activates $\mathcal{B}$ originates from $\mathcal{B}$ or one of its coordinatewise adjacent home blocks. There are at most 3^d such home blocks, each of which contributes at most ρ_c^X sender points. Consequently, the sender segment aligned with one receiver point contains at most $3^d \rho_c^X$ real candidate positions, and

$$C_{\text{Mercury}} \leq 3^d \rho_c^X N_Y.$$

For the current construction, $\rho_c^X, \rho_c^Y \leq 1$ gives $C_{\text{Mercury}} \leq 3^d N_Y$. Sender cell sparsity supplies the 3^d candidate bound, while receiver cell sparsity ensures that each receiver-active block represents one receiver point. Same-bin hash collisions may enlarge the padded point-table load, but block matching prevents entries with different block origins from becoming valid exact-test positions. This distinction explains why Table 1 contains both sender-table costs and a selected-candidate cost.

Density-parameterized spatial hashing. Piske et al. [43] require $\rho_c^X = 1$ and treat ρ_η^Y as a public capacity parameter. Each sender point is paired with at most ρ_η^Y receiver points from the relevant block neighborhood, which gives

$$C_{\text{PST}} \leq \rho_\eta^Y N_X.$$

A receiver point mapped to a target block must lie in that block or one of its coordinatewise adjacent home blocks. If Y is cell-sparse, these at most 3^d home blocks contribute at most one point each, so a public capacity of 3^d covers every receiver input admitted by **Mercury**. This is a common-domain worst-case setting rather than a claim that every such input realizes density 3^d . Their construction can support receiver inputs that are not cell-sparse by increasing ρ_η^Y and paying the corresponding multiplicative cost.

Minimum-separation conditions. For a point set Z , define

$$\Delta_{\min}^Z := \min_{\substack{z, z' \in Z \\ z \neq z'}} \|z - z'\|_2,$$

and take this minimum to be $+\infty$ when Z contains at most one point.

Receiver-side minimum separation (Y-sep.). Van Baarsen and Pu [2] and Dang et al. [14] assume

$$\Delta_{\min}^Y > 2\delta(\sqrt{d} + 1).$$

This condition imposes no corresponding separation requirement on X . For Dang et al. [14], Table 1 reports the variant that uses this receiver-separation bound and incurs the displayed 2^d factors. The separation bound ensures that each relevant grid region is associated with at most a constant number of receiver neighborhoods, so the induced candidate multiplicity is linear in the sender input size. It also implies receiver cell sparsity, but it constrains only Y , whereas the cell-sparsity condition of Mercury applies to both inputs.

Apart-input L_2 condition (apart). The construction of van Baarsen and Pu [56] assumes

$$\Delta_{\min}^X \geq 2\delta\sqrt{d} \quad \text{and} \quad \Delta_{\min}^Y \geq 2\delta(\sqrt{d} + 1).$$

The first bound ensures that a 2δ -side block contains at most one sender point, while the second ensures that such a block intersects at most one receiver ball of radius δ . Unlike the receiver-only condition above, the apart-input condition constrains both inputs.

Projection-separation conditions. For a point $z \in Z$ and coordinate ℓ , define the coordinatewise gap

$$g_Z(z, \ell) := \min_{z' \in Z \setminus \{z\}} |z[\ell] - z'[\ell]|,$$

and again take the minimum of an empty set to be $+\infty$. We say that coordinate ℓ separates z at threshold δ if $g_Z(z, \ell) > 2\delta$. Equivalently, the radius- δ projection interval of z on coordinate ℓ is disjoint from those of all other points in Z .

Grouped-coordinate s -separation. For $s \mid d$, partition the coordinates into the consecutive groups

$$G_j := \{(j-1)s + 1, \dots, js\}, \quad j \in [d/s].$$

An input set Z is s -separated if, for every $z \in Z$ and every group G_j , some coordinate $\ell \in G_j$ separates z . The case $s = 1$ requires every coordinate to separate every point, while $s = d$ requires one separating coordinate for each point and coincides with disjoint projection.

Protocol instantiations. Zhang et al. [62] require both X and Y to be 1-separated. The numeral in 1-sep. therefore denotes the coordinate-group size rather than a distance threshold. Gao et al. [20] and Yang et al. [59] require both inputs to satisfy disjoint projection. The separating coordinate may depend on the point, but these conditions prevent multiple same-party points from reusing the identifier that induces a cross-party candidate. Consequently, their identifier-induced candidate multiplicity remains linear in the input sizes. They also imply cell sparsity because two points in the same 2δ -side block differ by less than 2δ in every coordinate. The converse does not hold because cell sparsity imposes no coordinatewise separation across block boundaries. For example, the two-dimensional

set $\{(0, 0), (3\delta, 0), (0, 3\delta)\}$ is cell-sparse, but the point $(0, 0)$ has no coordinate that separates it from both remaining points. For the s -separation family, each point generates $O(\delta^s d/s)$ representations, so candidate multiplicity alone does not determine the cost. Table 1 therefore reports the $s = 1$ instantiation of Zhang et al. [62].

Routing and mini-universe conditions. The following assumptions constrain protocol routing or structured assignments rather than geometric block occupancy.

Disjoint hashing. Richardson et al. [46] limit the number of proximity-test instances routed to the same bin, although bounded sub-bin occupancy can relax this condition at additional routing cost. This is a routing constraint, whereas Mercury permits different block origins to collide in an ordinary hash bin and removes these collisions through private block matching.

Unique mini-universes. Structure-aware PSI [7] assigns the structured party’s balls to distinct active mini-universes while leaving the other party’s point set unrestricted, so its condition constrains a structured assignment rather than the point density of both inputs.

Impact of input assumptions. Under our criterion, an input assumption determines how much candidate reduction a coarse filter can guarantee. Without a restriction on local point concentration, a complete filter may retain all $N_X N_Y$ pairs, as in the baseline above. Cell sparsity bounds the occupancy of each grid block, while neighborhood density bounds the number of points mapped to a given block by spatial hashing. Minimum-separation conditions limit the overlap of geometric neighborhoods, and projection-separation conditions limit the reuse of filter identifiers. Routing and mini-universe conditions impose occupancy restrictions on the corresponding protocol instances. Although these assumptions constrain different objects, each controls how many cross-party pairs can survive the associated filter.

For Mercury, sender and receiver cell sparsity play distinct roles in establishing the filtered candidate structure. Sender cell sparsity limits each receiver block to at most 3^d candidate sender points, since only that block and its coordinatewise adjacent home blocks can contribute points to it. Receiver cell sparsity ensures that each receiver-active block contains a unique receiver point. Together, these conditions give $C_{\text{Mercury}} \leq \min\{N_X N_Y, 3^d N_Y\}$, so the number of selected logical candidates is linear in N_Y for fixed d . This bound applies after block alignment, whereas same-bin collisions and padding can add work at earlier stages.

E Standard FPSI

The standard variant Mercury returns

$$X_{\text{near}} := \{x \in X \mid \exists y \in Y : \text{dist}(x, y) \leq \delta\}$$

to the receiver and nothing to the sender. Unlike Mercury-SP, this variant opens individual candidate predicates before retrieving sender points. A cyclic shift

alone does not randomize the relative positions of these points within their sender segment. We therefore modify the sender point-table construction and hint interface before distance sharing, rather than only replacing the final output operation of Mercury-SP. The geometric activation, hash-table construction, encrypted distance evaluation, and secure masked-range test remain as defined in the main text.

E.1 Candidate Address Construction

After inserting the records of all sender blocks, the sender pads each bin to exactly C_{pt} slots and applies an independent uniform permutation to all its real and dummy slots. For block j in bin i , let $Q_{i,j} \subseteq [C_{pt}]$ be the positions occupied by its own real records, and let $t_{i,j} = |Q_{i,j}| \leq C_{act}$. Choose $C_{act} - t_{i,j}$ addresses uniformly without replacement from the entire complement $[C_{pt}] \setminus Q_{i,j}$. Writing this filler set as $U_{i,j}$, define

$$A_{i,j} = Q_{i,j} \cup U_{i,j} = \{a_{i,j,1} < \dots < a_{i,j,C_{act}}\}, \quad f[i][j][k] = \mathbf{1}[a_{i,j,k} \in Q_{i,j}].$$

Fillers may point to dummy slots or to real records owned by other blocks. They do not become records of block j , and selecting them never overwrites their owners' distance masks. The permutation applies only to the C_{pt} point-table positions in each bin, excluding any additional padding used for BFV batching.

The Non-SP hint interface gives the receiver the sorted address set of its matching block, instead of a segment-start index. It gives the parties secret shares of $eq[i]$ and $val[i][k] = eq[i] \wedge f[i][j][k]$. For a real receiver block with no match, the decoded address set is uniform among the C_{act} -subsets of $[C_{pt}]$, and every validity bit is zero. An empty receiver bin uses the local default $\{1, \dots, C_{act}\}$ and zero validity bits, without an OPRF query. The transferred descriptor is the combinatorial rank of $A_{i,j}$, protected by addition modulo $\binom{C_{pt}}{C_{act}}$ with a row-bound PRF pad. Ranking is a fixed bijection between the C_{act} -subsets of $[C_{pt}]$ and $\mathbb{Z}_{\binom{C_{pt}}{C_{act}}}$. Subtracting the pad and unranking therefore always gives a legal address set, and unused rank rows are uniform in the same domain.

Lemma 4. *For one block with $t \leq C_{act}$ real records, its candidate set A is a uniform C_{act} -subset of $[C_{pt}]$, independently of t . Conditional on A , any fixed μ distinct labeled records of that block occupy a uniform injection into A , independently of the number and values of its other records.*

Proof. The full-bin permutation makes the real address set Q a uniform t -subset of $[C_{pt}]$. For any fixed C_{act} -subset A_0 ,

$$\Pr[A = A_0] = \frac{\binom{C_{act}}{t}}{\binom{C_{pt}}{t} \binom{C_{pt}-t}{C_{act}-t}} = \frac{1}{\binom{C_{pt}}{C_{act}}}.$$

Given $A = A_0$, the set Q is a uniform t -subset of A_0 , and the t labeled records are assigned uniformly to its positions. For any prescribed injection of $\mu \leq t$

labeled records into A_0 , the number of completions is $\binom{C_{\text{act}} - \mu}{t - \mu} (t - \mu)!$. Dividing by the total $\binom{C_{\text{act}}}{t} t!$ gives

$$\frac{\binom{C_{\text{act}} - \mu}{t - \mu} (t - \mu)!}{\binom{C_{\text{act}}}{t} t!} = \frac{1}{C_{\text{act}} (C_{\text{act}} - 1) \cdots (C_{\text{act}} - \mu + 1)}.$$

This is the uniform-injection probability and contains no dependence on t . The empty injection covers $\mu = 0$.

The lemma concerns the single block addressed by the receiver in each bin. Address sets of different sender blocks in the same bin need not be independent. The complete proof below preserves their hidden ownership and simulates all exposed tables.

E.2 Protocol

The sender and receiver retain the full OPRF keys of their blocks. The same context-bound PRF domains protect hints and distances, but the address column now encodes a subset rank instead of the segment-start index used in *Mercury-SP*. For a received set $A_i = \{a_{i,1} < \cdots < a_{i,C_{\text{act}}}\}$, define $\text{pos}_i(k) = a_{i,k}$. The distance pad always uses the logical index k in this sorted set, not the point-table address $a_{i,k}$.

In Fig. 15, the sender initializes every slot mask $\text{mask}_{\text{out}}[i][a]$ independently and uniformly in $\mathbb{F}_p$. Only the block owning a real record replaces its slot mask, and filler addresses do not modify these masks. Let $V[i][a]$ denote the value at address a after the receiver decrypts the masked response and adds its squared-norm term for bin i . Empty receiver bins use their default addresses and zero local distance pads, giving $\llbracket \text{Dist}[i][k] \rrbracket_{\text{R}} = V[i][k]$.

The point encoding must represent every valid coordinate offset for the selected dimension and threshold. An all-zero OT word can encode a legitimate offset, so output validity is determined by the choice bit, not by whether the received word is zero. Restoring the block origin before deduplication is essential because the same sender point can appear relative to different receiver blocks.

Cost accounting. The standard variant uses the same mC_{pt} point-table slots and mC_{act} secure range tests as *Mercury-SP*. Its full-bin permutation takes $O(mC_{\text{pt}})$ local word operations in the fixed-dimensional model of Table 1. The protected address rank uses $\ell_A = \lceil \log_2 \binom{C_{\text{pt}}}{C_{\text{act}}} \rceil$ bits. With $O(C_{\text{pt}})$ auxiliary rows per bin, the address and validity columns occupy $O(mC_{\text{pt}}(\ell_A + C_{\text{act}}))$ bits. Point retrieval adds mC_{pt} OTs, each transferring an $O(d \log \delta)$ -bit point encoding. Candidate-address generation and combinatorial encoding are local computations. Sorting and deduplicating $t \leq mC_{\text{act}}$ retrieved records requires $O(t \log(t + 1))$ point comparisons and no further communication.

E.3 Correctness

Assume cell-sparse inputs, successful hashing and encoding, and the arithmetic conditions of Section 4.4. Consider a neighboring pair (x, y) . Theorem 1 places

Protocol Π_{Mercury}

1. **Tables and hints.** Build the block tables and abort if $\text{Succ}_{\text{hash}}$ fails. Permute all $\mathbf{C}_{\text{pt}}$ sender slots in each bin and construct $A_{i,j}$ and $f[i][j]$ as in Appendix E.1. Use the protected rank column to give the receiver A_i and both parties shares of $\text{val}[i][k]$.
2. **Owned-slot distance sharing.** Sample independent $\text{mask}_{in}[i][k] \leftarrow_{\$} \mathbb{F}_p$ for all i, k . For each real record of block j , set

$$\text{mask}_{out}[i][a_{i,j,k}] = \text{mask}_{in}[i][k] + D_{K_S[i][j]}(i, k).$$

Use the packed masks mask_{out} in Algorithm 2 and return its masked ciphertexts. For each nonempty receiver bin, set

$$\begin{aligned} \llbracket \text{Dist}[i][k] \rrbracket_S &= \text{mask}_{in}[i][k], \\ \llbracket \text{Dist}[i][k] \rrbracket_R &= V[i][a_{i,k}] - D_{K_R[i]}(i, k). \end{aligned}$$

3. **Candidate predicates.** Apply $\mathcal{F}_{\text{SMRT}}$ to the distance shares with $\tau = \delta^2$. Securely combine its result with $\text{val}[i][k]$ and reveal only the resulting $\text{near}[i][k]$ bits to the receiver.
4. **Point retrieval.** Set $c_i[a_{i,k}] = \text{near}[i][k]$ for $k \in [\mathbf{C}_{\text{act}}]$ and all other choices to zero. For every sender point-table slot, run OT with messages $(0, \text{EncodePoint}(\text{PT}_S[i][a]))$ and choice $c_i[a]$. The receiver decodes the selected offsets, adds $\text{BT}_R[i].ori$, and outputs the deduplicated points.

Fig. 15. Standard FPSI with full-bin permutation and fixed-size candidate address sets.

the entry for x in the sender block matching the receiver's origin and hash-function index in the bin containing y . The permutation preserves this record, and its address belongs to $Q_{i,j} \subseteq A_{i,j}$. The corresponding validity bit is therefore one. Both parties use the same address and the same index k in the sorted set, so

$$\llbracket \text{Dist}[i][k] \rrbracket_R - \llbracket \text{Dist}[i][k] \rrbracket_S = \text{Dist}(x, y).$$

The range test accepts, and the output OT returns the offset of x relative to $\text{home}(y).ori$. Adding that origin recovers x .

Every point in X_{near} is therefore retrieved at least once, up to the negligible errors assumed in Appendix C.3. Deduplication retains a single occurrence of each recovered sender point.

E.4 Security

Theorem 7. *On cell-sparse inputs and successful table construction, Π_{Mercury} realizes $\mathcal{F}_{\text{FPSI}}$ against static semi-honest adversaries under the requirements of Appendix C and secure OT, up to the stated error terms.*

Proof. As in Appendix C.3, we assume successful table construction and negligible sampling errors and failure probabilities. The circuit-privacy premise is the same as in Theorem 4, not an additional consequence of Lemma 4. Let $Z = X_{\text{near}}$ be the ideal receiver output. Each hybrid retains the preceding view except for the stated replacements and their dependent computations.

We first consider a corrupted sender and define the following hybrids.

- H_0^S . This is the sender’s real view, including its block and point tables, full-bin permutations, candidate sets, masks, encrypted query, and primitive transcripts.
- H_1^S . Replace the OPRF, 2PC, and OT transcripts by their simulations using the sender’s actual inputs and prescribed output shares. Retain the sender’s block keys, masks, Boolean shares, and OT message pairs consistently across these calls.
- H_2^S . Replace the receiver query in H_1^S by encryptions of zero with the same BFV key distribution and public query format. Recompute the sender’s encrypted response using this query and its unchanged local tables and masks. Keep the same sender inputs and shares in the simulated 2PC and OT calls.

The ideal-world sender simulator constructs its block and point tables, full-bin permutations, candidate sets, owned-slot masks, and PRF values honestly from X . It samples the prescribed Boolean output shares and generates the zero query and primitive transcripts as in H_2^S . The sender receives no result, so this simulator needs only X , the public parameters, and the declared failure behavior.

The sender hybrids are computationally indistinguishable for the following reasons.

- **H_0^S and H_1^S .** OPRF query privacy, secure 2PC, and OT security give $H_0^S \approx_c H_1^S$. The calls have public fixed dimensions and reveal neither the candidate predicates nor the receiver’s OT choices to the sender.
- **H_1^S and H_2^S .** BFV input privacy gives $H_1^S \approx_c H_2^S$, and the honestly generated state and uniform shares complete its ideal-world simulation.

For a corrupted receiver, we use the following hybrids.

- H_0^R . This is the receiver’s real view, including its query state, block keys, complete hint and distance responses, candidate address sets, opened predicates, and received OT messages.
- H_1^R . Replace the OPRF, 2PC, and OT transcripts by their simulations on the receiver’s actual inputs, shares, choices, and prescribed outputs. Replace the complete HE response by the circuit-privacy simulation of its actual masked plaintexts. Keep the query, BFV key pair, encryption randomness, block keys, candidate addresses, opened predicates, and chosen OT messages unchanged.
- H_2^R . Replace the hint and distance PRF values under receiver-unknown block keys with independent uniform values for distinct key-domain-input combinations. Preserve repeated evaluations and every value derived under a receiver-known key. Recompute the affected encodings, masked responses, receiver inputs, and simulated transcripts consistently with these replacements.

- H_3^R . Replace the generation of the candidate addresses, complete hint tables, masked response, predicates, and chosen OT messages by the following sampler using Y, Z . Retain the receiver’s tables, query state, BFV key pair, and known block keys.
- 1. For each nonempty receiver bin i , let $y_i \in Y$ be the original receiver point represented by its stored offset and compute

$$Z_i = \{x \in Z \mid \text{dist}(x, y_i) \leq \delta\}.$$

Sample a uniform C_{act} -subset $A_i \subseteq [C_{\text{pt}}]$ and assign the points of Z_i uniformly to distinct addresses in A_i . Set $\text{near}[i][k] = 1$ exactly when $a_{i,k}$ is assigned a point of Z_i . Sample the address sets and assignments independently across bins, retaining the same point value whenever an element of Z belongs to several sets Z_i . Empty receiver bins use the prescribed default addresses and all-zero predicates.

2. Generate each complete OKVS and all auxiliary rows using the sampler from Appendix C.1. For each nonempty bin, constrain decoding under the receiver’s block key to yield A_i , a random match seed, and a uniform masked validity word, with the address column storing the protected rank of A_i . For empty bins, sample the tables without a decoding constraint and keep the fixed defaults, without an OPRF query.
3. Sample all $V[i][a]$ independently and uniformly in $\mathbb{F}_p$ for $a \in [C_{\text{pt}}]$. Derive the receiver’s distance shares using A_i and its known-key distance pads, with zero pads for empty bins. Subtract the receiver’s squared-norm term from V and include independently protected padding for BFV slots outside the point table to obtain the complete plaintext response. Apply the HE simulator from H_1^R to this response and the retained receiver state.
4. Sample the receiver’s Boolean shares according to the component interfaces. Invoke the 2PC simulators using the sampled receiver inputs and shares, opening exactly the $\text{near}[i][k]$ values fixed in the first step.
5. Set $c_i[a_{i,k}] = \text{near}[i][k]$ and all other OT choices to zero. At each choice-one position, the selected message encodes $x - \text{home}(y_i).ori$ for the point $x \in Z_i$ assigned to that address. All choice-zero messages are zero. Invoke the receiver OT simulator from H_1^R on these messages and the length- mC_{pt} choice vector. Restore the original coordinates and deduplicate the returned points to obtain Z .

These operations define the ideal-world receiver simulator from Y, Z and the public parameters. It generates its query state honestly and samples its OPRF outputs as in the hint simulation, retaining one consistent set of block keys, addresses, masked values, and output points. No sender point outside Z , hidden block occupancy, or unaccepted candidate predicate is needed to perform the sampling.

We justify the receiver transitions as follows.

- H_0^R and H_1^R . OPRF, 2PC, and OT security together with the circuit-privacy premise give $H_0^R \approx_c H_1^R$.

- $\mathbf{H}_1^R$ and $\mathbf{H}_2^R$. Multi-user PRF security gives $\mathbf{H}_1^R \approx_c \mathbf{H}_2^R$, since all receiver-known keys and their evaluations remain unchanged.
- $\mathbf{H}_2^R$ and $\mathbf{H}_3^R$. We verify the receiver’s view in four steps.
 1. The set Z_i is exactly the set of points retrieved through the bin containing y_i , since every sender neighbor of y_i belongs to Z and activates $\text{home}(y_i)$. Lemma 4 gives the joint distribution of A_i and the addresses assigned to these points, independently of the number of other records in the matching block. When Z_i is empty, the uniform address distribution covers both a nonmatching block and a matching block with no accepted point. Independent per-bin permutations and filler sampling justify the independent assignments, while the same point values preserve repeated outputs across bins. Empty bins use the fixed defaults in both hybrids.
 2. The masked validity word is uniform because of the unrevealed vector $u[i]$, even after fixing A_i and the accepted positions. The whole-table argument in Appendix C.1 gives the required joint distribution of the complete OKVS and all auxiliary rows under the receiver-known block keys. This also covers additional lookups and row scans under those keys.
 3. By Section 4.1, at most one sender block per bin has a receiver-known key because each OPRF input includes the hash-function index. If present, this block uses a separate independent mask $\text{mask}_{in}[i][k]$ for each of its real records. Every other real record has an independent unknown-key pad in $\mathbf{H}_2^R$, which hides any reuse of $\text{mask}_{in}[i][k]$ across blocks, and dummy slots have independent masks. These masks are independent of the hint randomness, and the hint and distance pads use separate PRF domains. The entire vector V therefore has the sampled uniform distribution jointly with the hints, addresses, and opened predicates.
 4. For a returned point $x \in Z_i$ at address $a_{i,k}$, the receiver can compute its distance and infer

$$\text{mask}_{in}[i][k] = V[i][a_{i,k}] - D_{K_R[i]}(i, k) - \text{Dist}(x, y_i).$$

Hybrid $\mathbf{H}_3^R$ preserves this relation by using the same V , address, and point in the distance and OT simulations. Other records of the matching block use different masks $\text{mask}_{in}[i][k]$, while records owned by other blocks remain protected by unknown-key pads. Filler addresses do not alter this argument because they never overwrite the owner’s mask. Thus the simulation preserves the distances derivable from Y, Z without exposing other records through the recovered masks.

The receiver’s Boolean shares follow the component interfaces, and the sampled addresses and points determine its opened predicates and OT choices. Applying the same HE, 2PC, and OT simulators to these sampled values preserves the complete response, opened predicates, and chosen messages. Thus $\mathbf{H}_2^R$ and $\mathbf{H}_3^R$ induce the same distribution of the receiver’s view, up to the stated sampling errors and failure probabilities.

Auxiliary-input composition uses the same keys, addresses, shares, and output points throughout both hybrid sequences. Under the stated assumptions, the

accumulated distinguishing advantages and statistical errors of the OPRF, PRF, HE, 2PC, and OT hybrids are negligible. The final hybrids are generated by the two ideal-world simulators described above, completing the proof.