Versatile and Fast Location-Based Private Information Retrieval with Fully Homomorphic Encryption over the Torus

Joon Soo Yoo[1], Taeho Kim[2], and Ji Won Yoon[1]

[1]School of Cybersecurity, Korea University, Seoul, Republic of Korea

[2]Institute of ICT Planning and Evaluation (IITP), Daejeon, Republic of Korea

Abstract

Location-based services often require users to share sensitive locational data, raising privacy concerns due to potential misuse or exploitation by untrusted servers. In response, we present VeLoPIR, a versatile location-based private information retrieval (PIR) system designed to preserve user privacy while enabling efficient and scalable query processing. VeLoPIR introduces three operational modes—interval validation, coordinate validation, and identifier matching—that support a broad range of real-world applications, including information and emergency alerts. To enhance performance, VeLoPIR incorporates multi-level algorithmic optimizations with parallel structures, achieving significant scalability across both CPU and GPU platforms. We also provide formal security and privacy proofs, confirming the system's robustness under standard cryptographic assumptions. Extensive experiments on real-world datasets demonstrate that VeLoPIR achieves up to 11.55× speed-up over a prior baseline. The implementation of VeLoPIR is publicly available at `https://github.com/PrivStatBool/VeLoPIR`.

Acknowledgments

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (No. RS-2024-00460321, Development of Digital Asset Transaction Tracking Technology to Prevent Malicious Financial Conduct in the Digital Asset Market).

1 Introduction

In today's digital landscape, location-based services (LBS) have become deeply embedded in everyday life—helping users discover nearby amenities, receive traffic updates, or get localized alerts. While these services offer convenience, they often rely on collecting and storing users' precise geographic locations. This raises serious concerns about privacy. In many cases, the location data is gathered by large corporations or government agencies and may be repurposed or even sold to third parties, such as advertisers or data brokers.

Regulations such as the GDPR and national privacy laws are designed to prohibit unauthorized tracking of users' location data. However, numerous investigations have revealed that major companies have violated these protections in practice [1, 2]. Reports show that users' location histories have been used to infer sensitive personal information, enabling long-term tracking, profiling, and even unauthorized sales to third parties. As a representative example, in 2025, Google reached a $1.375 billion settlement with the state of Texas after being accused of secretly collecting users' geolocation data, biometric identifiers, and incognito search history without consent—one of the largest privacy settlements in U.S. history [3].

In conventional client-server communication, the reason the server can access user data is that both parties typically share a secret key established through a protocol such as Diffie-Hellman (DH) [4] or Elliptic Curve Diffie-Hellman (ECDH) [5]. While this ensures that data is encrypted during transmission, it does not prevent the server itself from decrypting and inspecting the data. As a result, although the server may correctly execute the user's requested query, it retains full access to the plaintext, creating opportunities for misuse or unauthorized data collection. This inherent trust assumption poses a significant privacy risk in scenarios where the server cannot be fully trusted.

A promising cryptographic solution to prevent unauthorized access by the server—while still allowing it to process user queries—is homomorphic encryption (HE) [6]. Unlike traditional encryption methods, HE does not require a key exchange protocol such as Diffie-Hellman, because it allows computations to be performed directly on encrypted data without ever decrypting it. As a result, HE ensures data remains secure not only in transit but also from the server itself. In response to this capability, extensive research has been conducted on private information retrieval (PIR) [7–11], with a particular focus on minimizing computational overhead for both the client and server while ensuring that the server processes queries without learning their content.

Our work, VeLoPIR, falls within the family of PIR protocols, and more specifically, addresses the problem of location-based PIR. In this setting, a client can query information from a server based on their location, while ensuring that their geographic coordinates remain completely hidden from the server. VeLoPIR is built on TFHE (Fully Homomorphic Encryption over the Torus) [12, 13], a logic-gate-based FHE scheme well-suited for shallow [1], non-linear circuits. Unlike multi-server [14], our single-

[1]In TFHE, "shallow circuits" refer to logic circuits with a relatively small number of sequential gates. For instance, VeLoPIR circuits typically have an estimated depth of around 40–80.

server design ensures strong privacy guarantees while minimizing the client's computational burden.

VeLoPIR introduces three operational modes—Interval Validation (IntV), Coordinate Validation (CoV), and Identifier Matching (IdM)—each designed to support different types of location-based queries under strong privacy guarantees. While a simplified version of IntV has been explored in earlier work [15], VeLoPIR generalizes the model into a modular framework, extends it with additional query modes (CoV and IdM), and introduces systematic algorithmic and parallel optimizations for scalability. Importantly, VeLoPIR also provides formal security proofs and experimental validation using large-scale, real-world datasets, which were not addressed in prior designs.

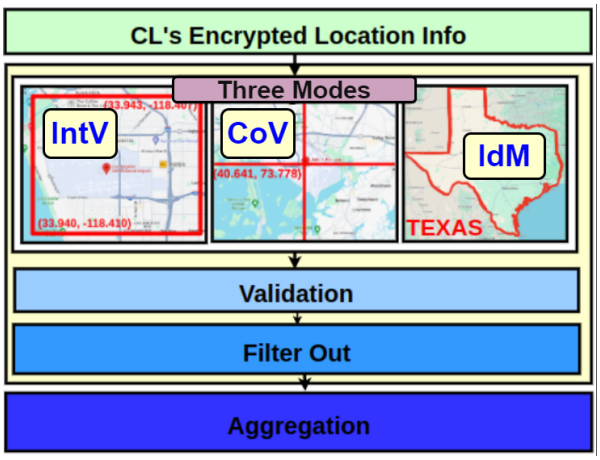


Figure 1: Overview of VeLoPIR's Structure.

For a simplified illustration of VeLoPIR, the server processes a client's (CL) encrypted locational data (see Fig. 1). At its core, VeLoPIR utilizes the operational modes IntV, CoV, and IdM as validation mechanisms for encrypted user locations, filtering out unrelated services. Each mode supports a distinct query type: IntV checks whether the location falls within a geographic interval, CoV validates exact coordinate matches, and IdM performs symbolic location matching based on identifiers such as city names or postal codes. The filtered result will be either the encryption of 0 or the encryption of the requested service, which is then aggregated to form the encrypted response. This response is returned to CL, who alone holds the secret key to decrypt the result. Throughout this process, the server has no access to the query result or any intermediate computations, thanks to FHE's capability for computation on encrypted data.

In summary, the contributions of our work are as follows:

- We introduce VeLoPIR, a versatile location-based PIR system that supports a broad range of real-world data types through three operational modes: IntV, CoV, and IdM, each preserving user locational privacy under different query structures.
- We design parallelized algorithms for VeLoPIR that exploit both CPU and GPU resources, enabling scalable and efficient evaluation over large datasets.
- We evaluate VeLoPIR using real-world datasets in both information and emergency alert scenarios, demonstrating significant improvements in query efficiency and practical applicability.
- We provide formal correctness proofs and conduct comprehensive security and privacy analysis, ensuring VeLoPIR's robustness in protecting location privacy.
- Our system achieves up to 11.55× speed-up over baseline approaches for location-specific queries on the `covid-usa` dataset, reducing the query time from 113.44 seconds to 9.83 seconds.[2]

1.1 Related Works

Our work can be viewed as a specialized application of private information retrieval (PIR) [7], focusing specifically on location-based queries. Traditional single-server PIR schemes often rely on server-side preprocessing [10, 11] to achieve sublinear query time. However, such preprocessing is tightly coupled to the structure of the database and must be repeated whenever updates occur, limiting flexibility. Other approaches [16] introduce an offline phase to generate cryptographic hints that reduce the client's online cost, but these methods still incur significant overhead during setup. More importantly, existing PIR schemes generally do not address the unique challenges of location-based retrieval. We summarize relevant location-based PIR methods in Table 1.

Lin et al. [17] focus on efficient PIR for large datasets using preprocessing and the BV scheme [18] from FHE. Their work primarily addresses the theoretical aspects of reducing computational complexity in general PIR, making it distinct from our VeLoPIR, which is optimized for location-specific queries in smaller, practical datasets.

Jain et al. [14] propose a location-based recommendation service using a dual-serve model, leveraging a hybrid encryption approach with ElGamal [20] and Paillier [19] schemes to preserve user privacy. Their system offloads the majority of computation to the two servers, thereby minimizing the client's computational burden. However, it relies on the assumption that the servers do not collude. As the dataset size increases, the recommendation time grows significantly, reaching 4355 seconds for 5000 elements.

An et al. [21] address location-based private information retrieval in a single-server setup for COVID-19 alerts using the BFV scheme [22]. While their approach enables proximity calculations on encrypted data, it imposes offline computational overhead on the user's device. Moreover, the patient's locational privacy is not preserved, as raw location data is shared with the authorities, who can also access the patient's contact history, potentially exposing auxiliary information; and the overall computation time is approximately 399 seconds.

[2]The baseline is derived from the prior LocPIR framework [15].

Table 1: Comparison of Location-Based PIR Schemes (PQ Secure indicates post-quantum security).

Scheme	Single Server	Offline (Client)	PQ Secure	Preprocessing	Location Specific	HE Scheme
Lin et al. [17]	✓	✗	✓	✓	✗	BV
Jain et al. [14]	✗	✗	✗	✓	✗	ElGamal/Paillier
An et al. [21]	✓	✓	✓	✓	✓	BFV
VeLoPIR **(Ours)**	✓	✗	✓	✗	✓	TFHE

2 Background

We use bold uppercase letters to denote matrices (e.g., $\mathbf{S}$) and bold lowercase letters to indicate vectors (e.g., $\mathbf{s}$). Scalars are represented by italic letters (e.g., s). When referring to the binary representation of a scalar, we use bracket notation (e.g., $s[i]$), where $s[i]$ denotes the i-th bit of the binary representation of the scalar s. A summary of the notations used is provided in Appendix **??** for reference.

2.1 Fully Homomorphic Encryption (FHE)

Homomorphic encryption enables computations on encrypted data without decryption, allowing a function f to be evaluated on encrypted inputs. Given ciphertexts $\mathsf{Enc}_{\mathsf{sk}}(x)$ and $\mathsf{Enc}_{\mathsf{sk}}(y)$, the evaluation algorithm Eval produces an encrypted result that, when decrypted, matches $f(x, y)$:

$$\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Eval}_f(\mathsf{Enc}_{\mathsf{sk}}(x), \mathsf{Enc}_{\mathsf{sk}}(y), \mathsf{evk})) = f(x, y).$$

Only the holder of the secret key sk can decrypt the result, while the encrypted inputs remain secure under the hardness of cryptographic assumptions, such as lattice-based problems.

Several FHE schemes exist, including quantum-resistant options NTRU [23] and Learning With Errors (LWE)-based schemes [24, 25], both of which rely on the hardness of lattice problems. Although LWE-based schemes introduce noise that limits the depth of evaluations, this limitation is overcome by Gentry's bootstrapping technique [26], which reduces noise, enabling further evaluations on the ciphertext. In our work, we employ the TFHE scheme [12], an LWE-based homomorphic encryption approach optimized for fast bootstrapping and efficient homomorphic logic gate evaluation. While arithmetic-based schemes like CKKS [27] are effective for low-depth arithmetic circuits, TFHE excels in handling shallow circuits and nonlinear operations, making it particularly suited for our work.

2.2 Torus Fully Homomorphic Encryption

TFHE is an LWE-based encryption scheme that operates over the torus $\mathbb{T} = \mathbb{R}/\mathbb{Z}$. It uses multiple types of ciphertexts (TLWE, TRLWE, and TRGSW) to enable efficient homomorphic operations, including the construction of logical gates. For the purpose of this paper, we specifically focus on the TLWE ciphertext $(\mathbf{a}, b)$.

Definition 1 (TLWE Problem)**.** *Let $n \in \mathbb{N}$ be a positive integer, and let $\mathbf{s} = (s_1, \ldots, s_n) \in \mathbb{B}^n$ be a secret vector where each s_i is sampled uniformly from the binary space $\mathbb{B} = \{0, 1\}$. Let χ be a Gaussian error distribution over the torus $\mathbb{T} = \mathbb{R}/\mathbb{Z}$. The Torus Learning with Errors (TLWE) problem is the task of distinguishing between samples drawn from the following two distributions:*

$$\mathcal{D}_0 = \{(\mathbf{a}, r) \mid \mathbf{a} \xleftarrow{\$} \mathbb{T}^n, r \xleftarrow{\$} \mathbb{T}\},$$
$$\mathcal{D}_1 = \left\{(\mathbf{a}, b) \mid \mathbf{a} \xleftarrow{\$} \mathbb{T}^n, b = \langle \mathbf{a}, \mathbf{s} \rangle + e \mod 1, e \leftarrow \chi\right\}.$$

Definition 2 (Advantage of an Adversary)**.** *The advantage of an adversary $\mathcal{A}$ in distinguishing between the two distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ in the TLWE problem is defined as:*

$$Adv^{\mathcal{A}}_{TLWE} = \Big| \Pr[\mathcal{A}(\mathbf{a}, r) = 1 \mid (\mathbf{a}, r) \leftarrow \mathcal{D}_1] - \Pr[\mathcal{A}(\mathbf{a}, r) = 1 \mid (\mathbf{a}, r) \leftarrow \mathcal{D}_0] \Big|.$$

The TLWE problem is said to be λ-secure if, for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, the advantage $\mathrm{Adv}^{\mathcal{A}}_{\mathrm{TLWE}}$ is at most $2^{-\lambda}$.

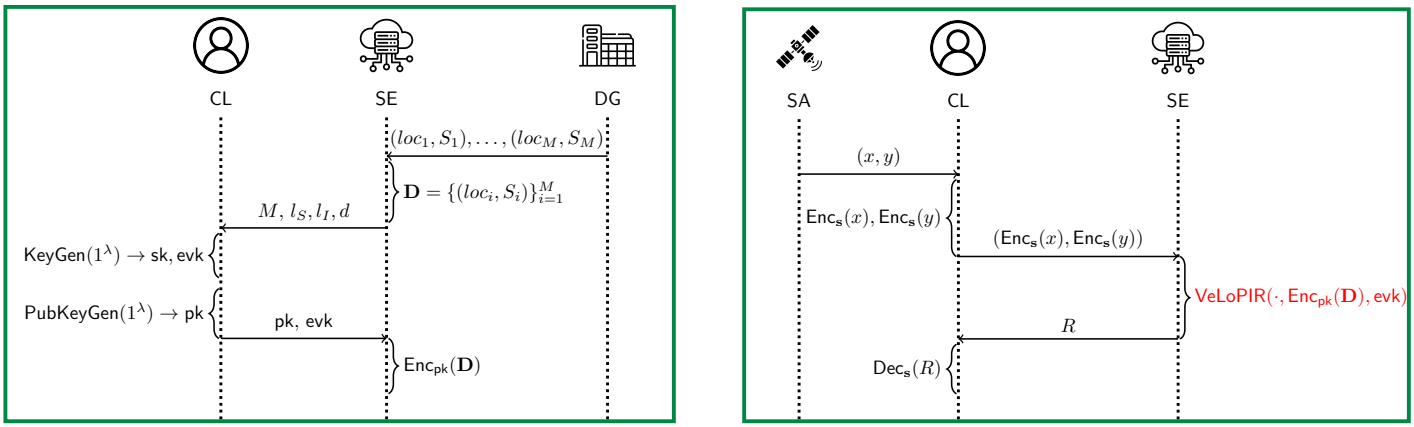
Based on the security of the TLWE ciphertext (similar to other TRLWE and TRGSW ciphertexts), the general TLWE encryption scheme is outlined as follows:

- $\mathsf{KeyGen}(1^\lambda)$: Given a security parameter λ, the key generation algorithm defines TLWE parameters n and σ, and outputs keys: a secret key $\mathbf{s} \xleftarrow{\$} \mathbb{B}^n$ and an evaluation key set evk, which includes a bootstrapping key and a key switching key.

- $\mathsf{Enc}_{\mathbf{s}}(m)$: Given a binary message $m \in \mathbb{B}$, it is encoded to a plaintext $\mu \in \mathbb{T}$ using the function $\mathsf{Ecd} : m \mapsto m/4 - 1/8$. Next, the algorithm generates a masking vector $\mathbf{a}$ by uniformly sampling from the set $U(\mathbb{T}^n)$. The plaintext message μ is then encrypted using the secret key $\mathbf{s}$, resulting in a TLWE sample denoted as $\mathsf{ct} = (\mathbf{a}, b)$ where $b = \langle \mathbf{a}, \mathbf{s} \rangle + \mu + e$. Here, e is a noise term drawn from a Gaussian distribution $N(0, \sigma)$.

- $\mathsf{Dec}_{\mathbf{s}}(\mathsf{ct})$: The decryption algorithm calculates the phase of the ciphertext $\mathsf{ct} = (\mathbf{a}, b)$ by $\varphi_{\mathbf{s}}(\mathsf{ct}) = b - \langle \mathbf{a}, \mathbf{s} \rangle$. The resulting phase $\mu + e$ is then rounded to the nearest plaintext message from the set $\{-1/8, 1/8\}$. Applying the inverse function of Ecd to the obtained plaintext message allows us to recover the original message bit $m \in \mathbb{B}$. Note that the error term e must satisfy $|e| < 1/16$ to ensure correct decryption.

TFHE logic gates are designed to refresh, or bootstrap, the noise with each logical operation. The bootstrapping ($\mathsf{Bootstrap}$) process in TFHE involves complex procedures such



(a) Preprocessing Phase

(b) Evaluation Phase

Figure 2: Preprocessing and Evaluation Phases of the VeLoPIR Protocol

as BlindRotate, SampleExtract, and KeySwitch. This bootstrapping phase typically consumes most of the time and memory during circuit evaluation, as it involves looking up elements in a precomputed TRLWE ciphertext table (see [12, 13] for more details).

The construction of logical gates in TFHE utilizes the bootstrapping technique as described previously. Specifically, we demonstrate the homomorphic evaluation of the AND operation as follows:

$$\mathsf{HomAND}(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{evk}) = \mathsf{Bootstrap}\left((\mathbf{0}, -\frac{1}{8}) + \mathsf{ct}_1 + \mathsf{ct}_2\right).$$

Assuming that the magnitude of the errors in both ct_1 and ct_2 is less than $1/16$, this procedure correctly produces the AND result. Similarly, we use HomXOR and HomXNOR to denote homomorphic XOR and XNOR operations, respectively.

3 Overall Protocol and Properties

The proposed protocol involves four parties: a client (CL), a server (SE), a data generator (DG), and a satellite (SA). The primary objective is to protect the client's locational data, ensuring that it can only be accessed by the client.

3.1 Assumptions

Semi-Honest Server. The server follows the protocol correctly but is considered curious and may attempt to learn the client's locational information (x, y).

Secure Client-Satellite Communication. The client securely obtains its locational information (x, y) from the satellite, ensuring confidentiality during transmission.

3.2 Detailed Protocol

3.2.1 Preprocessing Phase

The protocol involves three parties: CL, SE, and DG (see Fig. 2a).

1. **Data Collection.** SE collects data from DG to create a dataset $\mathbf{D} = \{(loc_i, S_i)\}_{i=1}^{M}$. Here, each loc_i represents a location, and S_i represents associated data or services.
2. **Secure Parameter Transmission.** SE sends the number of datasets M, data precision l_S, interval precision l_I, coordinate dimension d to CL using a secure channel (e.g., TLS/SSL).
3. **Key Generation by Client.** CL generates a secret key $\mathbf{s}$ and evaluation keys evk using the $\mathsf{KeyGen}(1^\lambda)$ algorithm. Additionally, CL performs the PubKeyGen algorithm to produce a public key set pk (refer to Algorithm 1). The generated public key set pk and evk are sent to SE.

Algorithm 1: $\mathsf{PubKeyGen}(1^\lambda, \mathbf{s}, M, l_I, l_S, d) \to \mathsf{pk}$

```
Initialize pk^I, pk^S ← ∅;
for i = 1 to M do
    for j ← 0 to l_I − 1 do
        for k ← 1 to d do
            a ←$ T^n; e ← χ; b ← ⟨a, s⟩ + e;
            ct_{i,j,k} ← (a, b);
            pk^I ← pk^I ∪ {ct_{i,j,k}};
    for j ← 0 to l_S − 1 do
        a ←$ T^n; e ← χ; b ← ⟨a, s⟩ + e;
        ct_{i,j} ← (a, b);
        pk^S ← pk^S ∪ {ct_{i,j}};
return pk = {pk^I, pk^S};
```

4. **Data Encryption by Server.** SE encrypts the dataset $\mathbf{D}$ using CL's public key pk through the ServerEnc process (refer to Algorithm 2).

3.2.2 Evaluation Phase

The evaluation protocol involves three parties: SA, CL, and SE (see Fig. 2b).

Algorithm 2: $\mathsf{ServerEnc}(\mathbf{D}, \mathsf{pk}) \rightarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{D})$

Initialize $\mathsf{Enc}_{\mathsf{pk}}(\mathbf{D}) \leftarrow \emptyset$;
for $i \leftarrow 1$ **to** M **do**
  for $j \leftarrow 0$ **to** $l_I - 1$ **do**
    for $k \leftarrow 1$ **to** d **do**
      $\mathsf{ct}_{i,j,k} \leftarrow (\mathbf{0}, \mathsf{Ecd}(x_{i,k}[j])) + \mathsf{pk}^I_{i,j,k}$;
      $\mathsf{Enc}_{\mathsf{pk}}(\mathbf{D}) \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{D}) \cup \{\mathsf{ct}_{i,j,k}\}$;
  for $j \leftarrow 0$ **to** $l_S - 1$ **do**
    $\mathsf{ct}_{i,j} \leftarrow (\mathbf{0}, \mathsf{Ecd}(S_i[j])) + \mathsf{pk}^S_{i,j}$;
    $\mathsf{Enc}_{\mathsf{pk}}(\mathbf{D}) \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{D}) \cup \{\mathsf{ct}_{i,j}\}$;
return $\mathsf{Enc}_{\mathsf{pk}}(\mathbf{D})$;

1. **Location Reception by Client.** CL receives its locational information (x, y) from SA, where x represents the latitude and y the longitude of CL's location coordinates.
2. **Encryption of Location.** CL encrypts (x, y) using its secret key $\mathbf{s}$ to obtain $(\mathsf{Enc}_{\mathbf{s}}(x), \mathsf{Enc}_{\mathbf{s}}(y))$. These encrypted coordinates are sent to S.
3. **Evaluation by Server.** SE uses the evaluation key evk to evaluate the VeLoPIR circuit given the encrypted GPS location $(\mathsf{Enc}_{\mathbf{s}}(x), \mathsf{Enc}_{\mathbf{s}}(y))$ of CL and its encrypted database $\mathsf{Enc}_{\mathsf{pk}}(\mathbf{D})$. The goal is to obtain the encrypted result R, which corresponds to the data associated with CL's location. Algorithm 3 in the following section provides the details of the server's evaluation process.
4. **Return of Encrypted Result.** SE sends the encrypted result R back to CL.
5. **Decryption by Client.** CL performs decryption using its secret key $\mathbf{s}$ to obtain the result: $\mathsf{Dec}_{\mathbf{s}}(R)$.

4 VeLoPIR Evaluation

VeLoPIR is structured around three operational modes: *Interval Validation*, *Coordinate Validation*, and *Identifier Matching*, with the latter being an extension of the coordinate-based approach.

- **Interval Validation (IntV)**: Checks if a client's encrypted location falls within a specific geographical interval, such as city boundaries.
- **Coordinate Validation (CoV)**: Verifies if the client's encrypted coordinates match a predefined target location.
- **Identifier Matching (IdM)**: Extends CoV by matching the client's encrypted location to an identifier (e.g., city name or postal code) instead of direct coordinates.

Note that instead of directly matching coordinates (x, y) (as in CoV), IdM maps these coordinates to a string identifier id_{CL}. Since IdM follows the same validation process as CoV, the correctness, security, and privacy proofs for CoV naturally extend to IdM, ensuring that identifier matching offers the same guarantees as coordinate validation.

4.1 Algorithm Overview

For each entry in the encrypted database $\mathsf{Enc}_{\mathsf{pk}}(\mathbf{D})$, which contains encrypted location coordinates loc_i and associated data S_i for up to M entries, the following steps are performed:

1. **Validation.** Determine whether the user's encrypted coordinates $(\mathsf{Enc}_{\mathbf{s}}(x), \mathsf{Enc}_{\mathbf{s}}(y))$ fall within loc_i. If a match is found, output $\mathsf{Enc}_{\mathbf{s}}(1)$; otherwise, output $\mathsf{Enc}_{\mathbf{s}}(0)$. Store the validation result as v. This step utilizes IntV, CoV, or IdM.
2. **Zero Out Unrelated Data.** Use the validation result v to eliminate unrelated services or data by performing a bitwise AND operation with all S_i. If validation is successful at position κ, only S_κ remains unchanged, while other S_i become vectors of encrypted zeros $\mathsf{Enc}_{\mathbf{s}}(0)$.

Aggregate Results. After the iteration of all locational entries, sum all S_i. Since only S_κ is non-zero, the result will be S_κ.

Algorithm 3: $\mathsf{VeLoPIR}(\mathsf{Enc}_{\mathbf{s}}(x), \mathsf{Enc}_{\mathbf{s}}(y), \mathsf{Enc}_{\mathsf{pk}}(\mathbf{D}), \mathsf{evk})$

for $i \leftarrow 1$ **to** M **do**
  Validation: Check if $(\mathsf{Enc}_{\mathbf{s}}(x), \mathsf{Enc}_{\mathbf{s}}(y))$ fall within or match loc_i;
  `/* Use IntV, CoV, or IdM */`
  if *validation is successful* **then**
    $v \leftarrow \mathsf{Enc}_{\mathbf{s}}(1)$;
  else
    $v \leftarrow \mathsf{Enc}_{\mathbf{s}}(0)$;
  Zero Out Unrelated Data: Apply v to filter the data S_κ;
  foreach S_i **do**
    $S_i \leftarrow \mathsf{HomBitwiseAND}(v, \mathsf{Enc}_{\mathbf{s}}(S_i), \mathsf{evk})$;
Aggregation: $R \leftarrow \mathsf{HomSum}(\{\mathsf{Enc}_{\mathbf{s}}(S_i)\}_{i=1}^{M}, \mathsf{evk})$;
return R;

4.2 Core Algorithms

(1) Interval Validation, IntV. For interval validation, the locational coordinates loc_i are defined by intervals of latitude (x_{left}, x_{right}) and longitude (y_{left}, y_{right}). The algorithm leverages nonlinear comparison operations, specifically less than or equal (HomCompLE) and less than (HomCompL), to perform these comparisons homomorphically (refer to Algorithm 4).

Given the encrypted client's locational information $(\mathsf{Enc}_{\mathbf{s}}(x), \mathsf{Enc}_{\mathbf{s}}(y))$, the algorithm IntV evaluates the following plaintext condition: $x_{left} \leq x < x_{right}$ and $y_{left} \leq y < y_{right}$. The validation outputs $\mathsf{Enc}_{\mathbf{s}}(1)$ if the conditions are satisfied; otherwise, it outputs $\mathsf{Enc}_{\mathbf{s}}(0)$.

Note that HomAND denotes the AND operation performed homomorphically on encrypted bits, producing the AND result. Similarly, HomXOR and HomXNOR represent homomorphic gates that perform XOR and XNOR operations, respectively (refer to Section 2 for additional background details).

Algorithm 4: $\mathsf{IntV}(\mathsf{Enc_s}(x), \mathsf{Enc_s}(y), \mathsf{Enc_s}(loc), \mathsf{evk})$

$/* \ \mathsf{Enc_s}(loc) = (\mathsf{Enc_s}(x_{left}), \ldots, \mathsf{Enc_s}(y_{right})) \ */$
$v_{x_{left}} \leftarrow \mathsf{HomCompLE}(\mathsf{Enc_s}(x_{left}), \mathsf{Enc_s}(x), l_I, \mathsf{evk})$;
$v_{x_{right}} \leftarrow \mathsf{HomCompL}(\mathsf{Enc_s}(x), \mathsf{Enc_s}(x_{right}), l_I, \mathsf{evk})$;
$v_{y_{left}} \leftarrow \mathsf{HomCompLE}(\mathsf{Enc_s}(y_{left}), \mathsf{Enc_s}(y), l_I, \mathsf{evk})$;
$v_{y_{right}} \leftarrow \mathsf{HomCompL}(\mathsf{Enc_s}(y), \mathsf{Enc_s}(y_{right}), l_I, \mathsf{evk})$;
$v_x \leftarrow \mathsf{HomAND}(v_{x_{left}}, v_{x_{right}}, \mathsf{evk})$;
$v_y \leftarrow \mathsf{HomAND}(v_{y_{left}}, v_{y_{right}}, \mathsf{evk})$;
$v \leftarrow \mathsf{HomAND}(v_x, v_y, \mathsf{evk})$;
return v;

We provide an in-depth discussion of the underlying algorithms, HomCompL and HomCompLE, which form the backbone of IntV, followed by the correctness of IntV.

Backbones of IntV: FHE Comparisons. HomCompLE is a non-linear FHE comparison circuit that takes two ciphertexts ct_1 and ct_2, and outputs $\mathsf{Enc_s}(1)$ if $x \leq y$, and $\mathsf{Enc_s}(0)$ otherwise. This operation is carried out using the evaluation key evk, with ct_1 and ct_2 being encryptions of z_1 and z_2 under the secret key $\mathbf{s}$ (refer to Algorithm 5).

Algorithm 5: $\mathsf{HomCompLE}(\mathsf{ct}_1, \mathsf{ct}_2, l_I, \mathsf{evk})$

$t_0 \leftarrow \mathsf{HomXOR}(\mathsf{ct}_1[l_I - 1], \mathsf{ct}_2[l_I - 1], \mathsf{evk})$;
$t_2 \leftarrow \mathsf{Enc}_{\mathbf{s}}^{\mathsf{TLWE}}(\mathbf{0}, \frac{1}{8})$
for $i \leftarrow 0$ **to** $l_I - 2$ **do**
  $t_1 \leftarrow \mathsf{HomXNOR}(\mathsf{ct}_1[i], \mathsf{ct}_2[i], \mathsf{evk})$;
  $t_2 \leftarrow \mathsf{HomMUX}(t_1, t_2, \mathsf{ct}_2[i], \mathsf{evk})$;
$r \leftarrow \mathsf{HomMUX}(t_0, \mathsf{ct}_1[l_I - 1], t_2, \mathsf{evk})$
return r

The circuit's design is fundamental to the correctness of the IntV algorithm, as discussed in [15]. For completeness, Theorem 1 formally establishes the correctness of HomCompLE, with the proof provided in Appendix A.1.

Theorem 1. *The homomorphic comparison* $\mathsf{HomCompLE}(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{evk})$ *stated in Algorithm 5 correctly evaluates whether* $z_1 \leq z_2$ *and outputs the result* r *as:*

$$r = \begin{cases} \mathsf{Enc_s}(1), & \text{if } z_1 \leq z_2 \\ \mathsf{Enc_s}(0), & \text{otherwise} \end{cases}$$

where ct_1 *and* ct_2 *are encryptions of* z_1 *and* z_2*, respectively, under the secret key* $\mathbf{s}$*.*

Proof. See Appendix A.1. □

HomCompL is similar in design to HomCompLE, with the only variation being the initialization of the t_2 variable, which is set to a trivial TLWE ciphertext of 0 (refer to Algorithm 6).

Corollary 1 establishes the correctness of Algorithm 6.

Corollary 1. *The homomorphic comparison* $\mathsf{HomCompL}(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{evk})$ *stated in Algorithm 6 correctly evaluates whether* $z_1 < z_2$ *and outputs the result* r *as:*

Algorithm 6: $\mathsf{HomCompL}(\mathsf{ct}_1, \mathsf{ct}_2, l_I, \mathsf{evk})$

$t_0 \leftarrow \mathsf{HomXOR}(\mathsf{ct}_1[l_I - 1], \mathsf{ct}_2[l_I - 1], \mathsf{evk})$;
$t_2 \leftarrow \mathsf{Enc}_{\mathbf{s}}^{\mathsf{TLWE}}(\mathbf{0}, -\frac{1}{8})$
for $i \leftarrow 0$ **to** $l_I - 2$ **do**
  $t_1 \leftarrow \mathsf{HomXNOR}(\mathsf{ct}_1[i], \mathsf{ct}_2[i], \mathsf{evk})$;
  $t_2 \leftarrow \mathsf{HomMUX}(t_1, t_2, \mathsf{ct}_2[i], \mathsf{evk})$;
$r \leftarrow \mathsf{HomMUX}(t_0, \mathsf{ct}_1[l_I - 1], t_2, \mathsf{evk})$;
return r;

$$r = \begin{cases} \mathsf{Enc_s}(1), & \text{if } z_1 < z_2 \\ \mathsf{Enc_s}(0), & \text{otherwise} \end{cases}$$

where ct_1 *and* ct_2 *are encryptions of* z_1 *and* z_2*, respectively, under the secret key* $\mathbf{s}$*.*

Proof. See Appendix A.2. □

We now formally establish the correctness of IntV, as illustrated in Algorithm 4, with the following theorem.

Theorem 2. *The interval validation algorithm (IntV) as stated in Algorithm 4 correctly evaluates whether the encrypted coordinate* $\mathsf{Enc_s}(x)$, $\mathsf{Enc_s}(y)$ *lies within the interval* $loc = (x_{left}, x_{right}, y_{left}, y_{right})$*. Specifically, it outputs:*

$$v \leftarrow \mathsf{Enc_s}(1)$$

if $x_{left} \leq x < x_{right}$ *and* $y_{left} \leq y < y_{right}$*, and otherwise outputs* $\mathsf{Enc_s}(0)$*.*

Proof. See Appendix A.4. □

(2) Coordinate Validation, CoV (and IdM). CoV efficiently matches the client's location coordinates (x, y) with a predefined service location loc_i. If the client's encrypted coordinates match loc_i, the corresponding service S_i is returned. CoV relies on the homomorphic equality comparison algorithm HomEQ (refer to Algorithm 7) for secure matching. Since IdM extends CoV by matching coordinates to an identifier, the correctness naturally follows from CoV, and we therefore focus only on CoV.

Algorithm 7: $\mathsf{HomEQ}(\mathsf{ct}_1, \mathsf{ct}_2, l_I, \mathsf{evk})$

$r \leftarrow \mathsf{Enc}_{\mathbf{s}}^{\mathsf{TLWE}}(\mathbf{0}, \frac{1}{8})$
for $i \leftarrow 0$ **to** $l_I - 1$ **do**
  $t \leftarrow \mathsf{HomXNOR}(\mathsf{ct}_1[i], \mathsf{ct}_2[i], \mathsf{evk})$;
  $r \leftarrow \mathsf{HomAND}(r, t, \mathsf{evk})$;
return r

The correctness of CoV (and IdM) relies on HomEQ, as established in Lemma 1, with the proof in Appendix A.3.

Lemma 1. *The homomorphic equality comparison* $\mathsf{HomEQ}(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{evk})$ *correctly evaluates whether* $z_1 = z_2$ *and outputs the result* r *as:*

$$r = \begin{cases} \mathsf{Enc_s}(1), & \text{if } z_1 = z_2 \\ \mathsf{Enc_s}(0), & \text{otherwise} \end{cases}$$

where ct_1 *and* ct_2 *are encryptions of* z_1 *and* z_2*, respectively, under the secret key* $\mathbf{s}$.

Proof. See Appendix A.3 □

CoV and IdM both utilize the HomEQ operation, as shown in Algorithm 8. In IdM, instead of applying homomorphic AND operation, HomEQ directly evaluates the encrypted identifier $\mathsf{Enc}_{\mathbf{s}}(id_{\mathsf{CL}})$.

Algorithm 8: $\mathsf{CoV}(\mathsf{Enc}_{\mathbf{s}}(x), \mathsf{Enc}_{\mathbf{s}}(y), \mathsf{Enc}_{\mathbf{s}}(loc), \mathsf{evk})$

```
/* (loc = (loc_x, loc_y)) */
v_x ← HomEQ(Enc_s(x), Enc_s(loc_x), evk);
v_y ← HomEQ(Enc_s(y), Enc_s(loc_y), evk);
v ← HomAND(v_x, v_y, evk);
return v;
```

The correctness of CoV (and IdM) is shown in Theorem 3.

Theorem 3. *The coordinate validation algorithm (CoV, similarly IdM), as stated in Algorithm 8, correctly evaluates whether the encrypted coordinates* $\mathsf{Enc}_{\mathbf{s}}(x)$ *and* $\mathsf{Enc}_{\mathbf{s}}(y)$ *match the encrypted location* $\mathsf{Enc}_{\mathbf{s}}(loc)$*. Specifically, it outputs:*

$$v \leftarrow \mathsf{Enc}_{\mathbf{s}}(1)$$

if $x = loc_x$ *and* $y = loc_y$*, and otherwise outputs* $\mathsf{Enc}_{\mathbf{s}}(0)$.

Proof. The correctness of the CoV algorithm can be directly inferred from the homomorphic equality checks performed by HomEQ. Specifically, the algorithm uses HomEQ to compare the encrypted coordinates $\mathsf{Enc}_{\mathbf{s}}(x)$ and $\mathsf{Enc}_{\mathbf{s}}(y)$ with $\mathsf{Enc}_{\mathbf{s}}(loc_x)$ and $\mathsf{Enc}_{\mathbf{s}}(loc_y)$ respectively.

For the final output v to be $\mathsf{Enc}_{\mathbf{s}}(1)$, both comparisons must return $\mathsf{Enc}_{\mathbf{s}}(1)$. This means that x must equal loc_x and y must equal loc_y. If either comparison fails, the result will be $\mathsf{Enc}_{\mathbf{s}}(0)$ due to the HomAND operation. □

4.3 Supporting Algorithms

(1) Zero-out Function. Once the validation result v is obtained, indicating whether the validation was successful ($\mathsf{Enc}_{\mathbf{s}}(1)$) or not ($\mathsf{Enc}_{\mathbf{s}}(0)$), this encrypted result can be used to zero out the service data S_i. The function $\mathsf{HomBitwiseAND}(v, \mathsf{Enc}_{\mathbf{s}}(S_i), \mathsf{evk})$ is applied to produce $\mathsf{Enc}_{\mathbf{s}}(S_\kappa)$, where S_κ is the desired locational information associated with loc_κ at index κ (refer to VeLoPIR Algorithm 3).

Algorithm 9: $\mathsf{HomBitwiseAND}(v, \mathsf{ct}, \mathsf{evk})$

```
/* l is the length of ct */
for i ← 1 to l do
    ct[i] ← HomAND(v, ct[i], evk);
return ct;
```

The HomBitwiseAND function ensures that $\mathsf{Enc}_{\mathbf{s}}(S_i)$ remains non-zero only when v is $\mathsf{Enc}_{\mathbf{s}}(1)$, achieved by applying the HomAND gate bitwise (see Algorithm 9). If v is $\mathsf{Enc}_{\mathbf{s}}(0)$, then $\mathsf{Enc}_{\mathbf{s}}(S_i)$ is zeroed out, meaning all bits are $\mathsf{Enc}_{\mathbf{s}}(0)$ due to the HomAND operation. This is summarized in Lemma 2, with the proof omitted.

Lemma 2. *The homomorphic function* HomBitwiseAND *in Algorithm 9 outputs* ct *if* $v = \mathsf{Enc}_{\mathbf{s}}(1)$*, and outputs* $\mathsf{Enc}_{\mathbf{s}}(0)$ *for all bits of* ct *if* $v = \mathsf{Enc}_{\mathbf{s}}(0)$*. Given the encrypted single bit* v*, the function effectively zeroes out the encrypted vector* ct *based on the value of* v.

(2) Aggregation. At the final stage of the VeLoPIR algorithm, we aggregate the results of $\mathsf{Enc}_{\mathbf{s}}(S_i)$ that have been validated by the encrypted bit v. Assuming no loc_i intersects with another loc_j for $i \neq j$, the resulting set $\{\mathsf{Enc}_{\mathbf{s}}(S_i)\}_{i=1}^{M}$ will contain only one non-zero element, $\mathsf{Enc}_{\mathbf{s}}(S_\kappa)$, while the other $\mathsf{Enc}_{\mathbf{s}}(S_i)$ values will be $\mathsf{Enc}_{\mathbf{s}}(0)$ across all l_S bits.

Algorithm 10: $\mathsf{HomSum}(\{\mathsf{Enc}_{\mathbf{s}}(S_i)\}_{i=1}^{M}, \mathsf{evk})$

```
/* Initialize S as a vector of
   encrypted zeros of length l_S */
S ← [Enc_s(0), ..., Enc_s(0)]
for i ← 1 to M do
    for j ← 0 to l_S − 1 do
        S[j] ← HomXOR(S[j], Enc_s(S_i[j]), evk);
return S;
```

Since all $\mathsf{Enc}_{\mathbf{s}}(S_i)$ values are encrypted, the server cannot distinguish between the encryption of zeros and $\mathsf{Enc}_{\mathbf{s}}(S_\kappa)$. Therefore, the server cannot naively select S_κ but must instead consider all possible cases. This is where homomorphic XORing comes into play through the HomSum algorithm, which ensures that the correct S_κ is obtained without revealing any unnecessary information (see Algorithm 10). The proof of Lemma 3 explains the correctness and necessity of the HomSum operation.

Lemma 3. *The homomorphic summation* HomSum *in Algorithm 10 over a set* $\{\mathsf{Enc}_{\mathbf{s}}(S_i)\}_{i=1}^{M}$ *correctly outputs* $\mathsf{Enc}_{\mathbf{s}}(S_\kappa)$*, where the set* $\{\mathsf{Enc}_{\mathbf{s}}(S_i)\}_{i=1}^{M}$ *contains only one non-zero* $\mathsf{Enc}_{\mathbf{s}}(S_\kappa)$*, and the rest* $\mathsf{Enc}_{\mathbf{s}}(S_i)$ *are encryptions of zeros across all* l_S *bits.*

Proof. Given that all $\mathsf{Enc}_{\mathbf{s}}(S_i)$ except $\mathsf{Enc}_{\mathbf{s}}(S_\kappa)$ are encryptions of zero across all l_S bits, the homomorphic XOR operation effectively acts as an addition. Specifically, the iterative operation $S[j] \leftarrow \mathsf{HomXOR}(S[j], \mathsf{Enc}_{\mathbf{s}}(S_i[j]), \mathsf{evk})$ simplifies to:

$$S[j] \leftarrow \mathsf{HomXOR}(\mathsf{Enc}_{\mathbf{s}}(0), \mathsf{Enc}_{\mathbf{s}}(S_\kappa[j]), \mathsf{evk})$$

Since XORing any bit with 0 yields the original bit, the operation correctly outputs $\mathsf{Enc}_{\mathbf{s}}(S_\kappa)$ for S in Algorithm 10. □

4.4 Correctness of VeLoPIR

We can now conclude that the VeLoPIR circuit correctly evaluates and retrieves the desired locational information $R = \mathsf{Enc}_{\mathbf{s}}(S_\kappa)$, corresponding to the encrypted coordinates $(\mathsf{Enc}_{\mathbf{s}}(x), \mathsf{Enc}_{\mathbf{s}}(y))$, as established through the series of preceding theorems and lemmas, culminating in Theorem 4.

Theorem 4. *The homomorphic location-based information retrieval algorithm,* VeLoPIR*, as described in Algorithm 3, correctly evaluates and retrieves the encrypted service data* $\mathsf{Enc}_{\mathbf{s}}(S_\kappa)$ *corresponding to the encrypted locational coordinates* $(\mathsf{Enc}_{\mathbf{s}}(x), \mathsf{Enc}_{\mathbf{s}}(y))$ *using the evaluation key* evk*.*

Proof. First, by Theorems 2 and 3, the validation step in VeLoPIR correctly evaluates whether the encrypted coordinates $\mathsf{Enc}_{\mathbf{s}}(x)$ and $\mathsf{Enc}_{\mathbf{s}}(y)$ match or fall within the location loc_κ. The result of this validation is an encrypted bit v that equals $\mathsf{Enc}_{\mathbf{s}}(1)$ if the coordinates match and $\mathsf{Enc}_{\mathbf{s}}(0)$ otherwise. Next, using Lemma 2, the HomBitwiseAND function applies this validation result v to each encrypted service data $\mathsf{Enc}_{\mathbf{s}}(S_i)$. This operation zeroes out all $\mathsf{Enc}_{\mathbf{s}}(S_i)$ except for $\mathsf{Enc}_{\mathbf{s}}(S_\kappa)$, which corresponds to the validated location loc_κ. Finally, by Lemma 3, the HomSum operation correctly aggregates the results to produce $R = \mathsf{Enc}_{\mathbf{s}}(S_\kappa)$. This ensures that only the service data corresponding to the validated location is returned.

□

5 Security and Privacy Analysis

Security and Privacy of VeLoPIR. We propose that the VeLoPIR system is both secure and private. This is because all operations are performed on encrypted data, with no decryption occurring during the evaluation process. The following theorem formalizes the security and privacy of our system.

Theorem 5. *Assuming that the size of the database* M *and the precision parameters* l_I *and* l_S *are bounded by* $O(\lambda^k)$ *for some constant* k*, the proposed* ***VeLoPIR*** *system is* λ-***secure*** *and preserves the* ***privacy*** *of the client's location.*

Proof. (Security) Given that the size of the database M and the precision parameters l_I and l_S are bounded by $O(\lambda^k)$ for some constant k, the adversary's advantage $\mathrm{Adv}^{\mathcal{A}}_{\mathsf{TLWE}_{1\text{ to }W_s}}$ remains negligible:

$$\mathrm{Adv}^{\mathcal{A}}_{\mathsf{TLWE}_{1\text{ to }W_s}} = \sum_{i=1}^{W} \mathrm{Adv}^{\mathcal{A}}_{\mathsf{TLWE}_i} < \frac{O(W_s)}{2^\lambda}$$

where $W_s = O(Ml_Il_S)$ denotes the number of TLWE samples occurring throughout the VeLoPIR system. Therefore, the system is λ-secure.

(Privacy) Since the input locational information $(\mathsf{Enc}_{\mathbf{s}}(x), \mathsf{Enc}_{\mathbf{s}}(y))$of the client is encrypted, and all intermediate results as well as the final output $R = \mathsf{Enc}_{\mathbf{s}}(S_\kappa)$ are encrypted under the client's secret key, no information about the client is leaked except for auxiliary information such as M, l_S, l_I, and d. Thus, the privacy of the user's location is preserved.

□

Threat 1: Auxiliary Information Leakage. One potential threat to the VeLoPIR protocol arises if the server obtains auxiliary information. This information could include the number of data entries M, the service precision l_S, the interval precision l_I, and the coordinate dimension d from the client data. However, even if the adversary (which could be the server or a third party intercepting the communication between the client and server) gains access to these auxiliary TLWE samples, the adversary's ability to break the TLWE encryption and retrieve the secret key $\mathbf{s}$ remains negligible.

Theorem 6. *The advantage of an adversary* $\mathcal{A}$*, who has access to auxiliary public keys* pk *and information such as* M*,* l_S*,* l_I*, and* d*, is negligible. Specifically, for a polynomial number of samples* $W = Ml_Id + Ml_S$*, the advantage is bounded by:*

$$Adv^{\mathcal{A}}_{\mathsf{TLWE}_{1\text{ to }W}} < \frac{O(W)}{2^\lambda}$$

where $W = Ml_Id + Ml_S$*.*

Proof. Given that the TLWE sample is λ-secure, the adversary's advantage for each individual sample is bounded by $2^{-\lambda}$. The total advantage across W samples is:

$$\mathrm{Adv}^{\mathcal{A}}_{\mathsf{TLWE}_{1\text{ to }W}} = \sum_{i=1}^{W} \mathrm{Adv}^{\mathcal{A}}_{\mathsf{TLWE}_i} < \frac{O(W)}{2^\lambda}$$

Since $W = Ml_Id + Ml_S$ grows polynomially with the input size, the adversary's total advantage remains negligible, ensuring the security of the protocol. □

Threat 2: Server Access to Raw Data. Another potential threat arises when the server has raw access to the dataset $\mathbf{D} = \{(loc_i, S_i)\}_{i=1}^{M}$, which contains service data associated with specific locational coordinates. This access is necessary for the server to provide the relevant services to the client, as it aggregates this information from local data generators. However, the primary objective of the VeLoPIR protocol is to protect the user's location, not the service data S_i itself (which the server already possesses). Consequently, the ability of the adversary (in this case, the server) to solve the TLWE sample and obtain the user's secret key $\mathbf{s}$ remains negligible. Lemma 2 directly follows from Theorem 6.

Corollary 2. *The advantage of the server in the* ***VeLoPIR*** *protocol, given access to the service data* $\mathbf{D} = \{(loc_i, S_i)\}_{i=1}^{M}$*, is negligible:*

$$Adv^{\mathcal{A}}_{\mathsf{TLWE}_{1\text{ to }W}} < \frac{O(W)}{2^\lambda}$$

where $W = Ml_Id + Ml_S$*, as defined in Theorem 6.*

Proof. See Appendix B.1. □

Threat 3: Insecure Communication Channel from Satellite to Client. A potential threat to the VeLoPIR protocol is the vulnerability of GPS information during transmission (refer to Figure 2b) from the satellite (SA) to the client (Cl), such as in GPS spoofing [29, 30] and jamming attacks [31, 32]. These attacks could alter the user's location before it is encrypted and processed by VeLoPIR, potentially rendering the retrieved information inaccurate. However, this threat is beyond the scope of the VeLoPIR protocol, which focuses on protecting the user's location data post-encryption. VeLoPIR ensures that location-based queries are securely processed without revealing the user's location to the server or any other party during the evaluation process.

Securing the communication channel between the satellite and the client is an ongoing research focus, particularly within the

realm of Post-Quantum Cryptography (PQC). Efforts are being made to establish a secure network using PQC algorithms [33–35], including Lattice-based cryptographic schemes like those based on the LWE problem. Given that the VeLoPIR protocol is also built on LWE, implementing LWE-based PQC schemes for secure satellite-to-client communication would not only enhance security but also ensure seamless integration with VeLoPIR.

6 VeLoPIR Optimization

Bit-Level Parallel Acceleration. In logic-based FHE schemes like TFHE, homomorphic gate operations, such as HomAND, are significantly slower than their plaintext equivalents. For example, evaluating a homomorphic AND gate, which involves steps like BlindRotate, SampleExtract, and KeySwitch, takes around 13 ms, whereas a plaintext gate takes only 0.3 ns—making the homomorphic operation about 40 million times slower. This considerable overhead makes TFHE well-suited for parallel optimization, as CPU and GPU parallelism can substantially improve performance. We leverage this observation to optimize the core algorithms in VeLoPIR.

6.1 Core Functionality Optimizations

The VeLoPIR circuit can be optimized at multiple levels, ranging from bitwise operations to high-level structural enhancements. We categorize these into three levels of parallel optimization—**Innermost**, **Mid-Level**, and **Outermost**—corresponding to the hierarchical structure of computation within the system (refer to Table 2 for a summary of these optimization levels and the associated components).

(1) Innermost Optimization: Nonlinear HE Operations. Optimizing nonlinear HE operations like HomCompLE, HomCompL, and HomEQ can significantly enhance the performance of the VeLoPIR system.

Homomorphic Comparsions. In the HomCompLE algorithm (Algorithm 5), the following loop is key:

$$t_1 \leftarrow \mathsf{HomXNOR}(\mathsf{ct}_1[i], \mathsf{ct}_2[i], \mathsf{evk})$$
$$t_2 \leftarrow \mathsf{HomMUX}(t_1, t_2, \mathsf{ct}_2[i], \mathsf{evk})$$

Here, t_2 is sequentially updated based on the result of the previous t_1 value. This sequential dependency means that the t_2 updates cannot be parallelized. However, the HomXNOR operations for each bit position are independent and can be executed in parallel using parallel processing units (see Algorithm 11).

Similarly, HomCompL can be optimized in the same manner as HomCompLE, improving performance by parallelizing the independent HomXNOR operations. Both HomCompL and HomCompLE utilize parallel resources to their maximum capacity, up to l_I units.

Homomorphic Equality. The HomEQ algorithm contains no dependencies between the outcomes, allowing parallel computation of HomXNOR on the input ciphertexts $\mathsf{ct}_1[i]$ and $\mathsf{ct}_2[i]$ up to l_I, producing $t_1[i]$ values. These values can then be reduced in parallel using pairwise HomAND operations, reaching a depth of $\log(l_I)$. The HomEqOPT algorithm can utilize up to l_I parallel processing units.

Algorithm 11: $\mathsf{HomCompLeOPT}(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{evk}, n_p)$

```
t_0 ← HomXOR(ct_1[l_I − 1], ct_2[l_I − 1], evk);
t_1 ← Enc_s^TLWE(0, 1/8);
foreach i ← 0 to l_I − 2 do
    /* allocate parallel processing
       units                                   */
    t_XNOR[i] ← HomXNOR(ct_1[i], ct_2[i], evk);
for i ← 0 to l_I − 2 do
    t_1 ← HomMUX(t_XNOR[i], t_1, ct_2[i], evk);
r ← HomMUX(t_0, ct_1[l_I − 1], t_1, evk);
return r;
```

Algorithm 12: $\mathsf{HomEqOPT}(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{evk}, n_p)$

```
foreach i ← 0 to l_I − 1 do
    /* allocate parallel processing
       units                                   */
    t_1[i] ← HomXNOR(ct_1[i], ct_2[i], evk);
/* Parallel reduction                         */
for k ← 1 to l_I by ×2 do
    for i ← 0 to l_I − 1 by 2k do
        if i + k < l_I then
            t_1[i] ← HomAND(t_1[i], t_1[i + k], evk);
return r ← t_1[0];
```

(2) Mid-Level Optimization: HomBitwiseAND. A straightforward optimization approach involves allocating parallel resources to the HomAND gate evaluation in Algorithm 9:

$$\mathsf{ct}[i] \leftarrow \mathsf{HomAND}(v, \mathsf{ct}[i], \mathsf{evk}). \tag{1}$$

Since VeLoPIR requires the evaluation of $\mathsf{HomBitwiseAND}(v, \mathsf{Enc}_{\mathbf{s}}(S_i), \mathsf{evk})$, this operation necessitates l_S parallel resources for maximum parallelization.

(3) Mid-Level Optimization: Validation Modules (IntV, CoV). *Interval Validation.* Validation in IntV can be optimized by parallelizing the computation of independent variables $v_{x_{left}}, v_{x_{right}}, v_{y_{left}}$, and $v_{y_{right}}$. These variables, which utilize similar operations to HomCompLE and HomCompL, can be computed simultaneously. We can parallel process the following operations line by line:

$$v_{x_{left}} \leftarrow \mathsf{HomCompLE}(\mathsf{Enc}_{\mathbf{s}}(x_{left}), \mathsf{Enc}_{\mathbf{s}}(x), \mathsf{evk})$$
$$v_{x_{right}} \leftarrow \mathsf{HomCompL}(\mathsf{Enc}_{\mathbf{s}}(x), \mathsf{Enc}_{\mathbf{s}}(x_{right}), \mathsf{evk})$$
$$v_{y_{left}} \leftarrow \mathsf{HomCompLE}(\mathsf{Enc}_{\mathbf{s}}(y_{left}), \mathsf{Enc}_{\mathbf{s}}(y), \mathsf{evk})$$
$$v_{y_{right}} \leftarrow \mathsf{HomCompL}(\mathsf{Enc}_{\mathbf{s}}(y), \mathsf{Enc}_{\mathbf{s}}(y_{right}), \mathsf{evk})$$

Additionally, v_x and v_y can be computed independently afterward. The algorithm can be parallelized using up to $2d$ resources.

Coordinate Validation. CoV can also benefit from parallel processing by pairing the independent v_x and v_y operations, both of

which use the HomEQ function. This allows the use of up to d parallel processing units.

(4) Outermost Optimization: Large-Scale Evaluation. The VeLoPIR evaluation can be optimized by leveraging M parallel processors to handle the large-scale processing unit, specifically the evaluation of each S_i in the VeLoPIR circuit. Since each S_i operates independently and requires similar computational effort, parallelization ensures efficient processing. The optimization can be designed as follows:

```
/* allocate parallel processing units   */
foreach i ← 0 to M do
    Validation v & Zero Out Unrelated Data:
    foreach S_i do
        S_i ← HomBitwiseAND(v, Enc_s(S_i), evk)
```

(5) Outermost Optimization: Aggregation via HomSum. The HomSum algorithm can be optimized using two approaches: parallel reduction and Bit-AND optimization. First, we apply pairwise evaluation of the HomXOR operation on the encrypted service data $\mathsf{Enc_s}(S_i[j])$ using parallel reduction across M processing units. This reduces the number of operations logarithmically with respect to M.

Next, for each pairwise HomXOR evaluation, we apply Bit-AND optimization, utilizing up to l_S parallel units for the operation:

$$\mathsf{Enc_s}(S_i[j]) \leftarrow \mathsf{HomXOR}(\mathsf{Enc_s}(S_i[j]), \mathsf{Enc_s}(S_{i+k}[j]), \mathsf{evk})$$

where k increases from 1 to $\log_2(M)$ during pairwise evaluation. Thus, the maximum parallel processing units required for HomSum optimization is $(M \cdot l_S)/2$.

7 Theoretical Resource Complexity

Parallel Processing Resource Complexity. In VeLoPIR, we utilize parallel processing to enhance algorithm efficiency. The resource complexity is computed by considering the hierarchical structure of the algorithms, such as HomCompLE and HomCompL within IntV. Table 2 outlines the maximum number of parallel units required for each component. The overall parallel processing resource complexity simplifies to:

$$O(M \cdot (l_S + d \cdot l_I))$$

This accounts for the dominant factors affecting parallel execution across the algorithm's components.

Theoretical Speed-up Bound. Theoretically, the maximum possible speed-up for VeLoPIR is $O(M \cdot (l_S + d \cdot l_I))$ as with the parallel processing resource complexity. This bound assumes that all parts of the algorithm can be fully parallelized without overhead. However, as we will show in the evaluation section, the actual performance is affected by factors such as communication overhead between the GPU and CPU. This limits the real-world speed-up compared to the theoretical maximum.

8 Application Scenarios

We explore three privacy-preserving strategies for private information alerts and location-based services: IntV, CoV, and IdM.

Table 2: Parallel resource complexity and hierarchy in VeLoPIR. Levels refer to the scope of parallelism: **Outermost** (across records), **Mid-Level** (across validation blocks), and **Innermost** (within comparison primitives).

Algorithm	Processing Units (n_p)	Parallel Level
Large-Scale	M	Outermost
HomSum	$\frac{Ml_S}{2}$	Outermost
HomBitwiseAND	l_S	Mid-Level
IntV	$2d$	Mid-Level
CoV / IdM	$d\,(1)$	Mid-Level
HomCompLE / HomCompL	l_I	Innermost
HomEQ	l_I	Innermost

(1) Bounding Box Validation (IntV). Bounding box validation is an efficient strategy when the user's location needs to be verified within a certain geographic boundary without revealing their exact position. This is particularly relevant in scenarios like pandemic disease alerts, where the client is interested in receiving alerts about nearby confirmed cases without explicitly disclosing their location. We demonstrate the performance of the IntV approach using datasets such as `covid-usa` and `covid-kor`.

(2) Exact Coordinate Validation (CoV). In certain cases, the user needs highly precise information about their exact geographic position. This approach is essential for applications like crime alerts or disaster management, where precise location-based notifications are crucial for immediate response. We use the `gdacs` dataset to demonstrate the efficiency of CoV in these situations.

(3) Identifier Matching (IdM). In many modern datasets, especially those related to healthcare or meteorological data, information is often categorized by administrative regions such as cities or states. In these cases, validating a user's location based on an identifier (e.g., city or state name) can be more efficient and relevant than using geographic coordinates. IdM leverages this structure to provide efficient and privacy-preserving location matching. We showcase the performance of IdM with the `covid-usa`, `covid-kor`, and `weather-us` datasets.

9 Evaluation

We evaluate VeLoPIR with the following key questions:

- **RQ1. Parallelization.** Can the core components of VeLoPIR—including outermost, mid-level, and innermost levels of parallelism—be efficiently parallelized across CPU cores and GPUs? Which hardware platform (CPU or GPU) is optimal for the VeLoPIR circuit?
- **RQ2. Efficiency.** How much speed-up can be achieved through our core optimizations across different datasets?
- **RQ3. Applicability.** Which operational mode (IntV, CoV, or IdM) is best suited for each real-world application scenario?

Reference Benchmark. In this work, we use LocPIR [15] as a reference benchmark for comparison. We evaluate VeLoPIR

to demonstrate its extended functionality and optimized performance for practical applications. All evaluations are conducted using real-world datasets (summarized in Table 3) to ensure that the improvements in VeLoPIR are measured against practical, relevant data.

9.1 Environment Setup

Hardware. Experiments were conducted on a 13th Gen Intel Core i9-13900K processor (24 cores, 32 threads, 5.8 GHz max frequency). The system operated on Ubuntu 24.04 LTS, utilizing TFHE library version 1.1. For GPU parallel processing, the system was equipped with an NVIDIA GeForce RTX 4060 Ti GPU (16 GB GDDR6 memory) running CUDA version 12.4.

TFHE Parameters. Our model employed a standard security level of 128-bit (λ_{128}) as in [13] (for details, see Appendix C.1).

Dataset (refer to Tab. 3). The `covid-kor` dataset [37], provided by the Korea Disease Control and Prevention Agency (KDCA), contains daily records of COVID-19 patient incidences across nine major cities in Korea as of October 26, 2021. The `covid-usa` dataset [36] contains daily confirmed COVID-19 cases across U.S. states, recorded on February 11, 2023. The `gdacs` dataset [38] is sourced from the Global Disaster Alert and Coordination System (GDACS), capturing disaster information (e.g., earthquakes, tsunamis, etc.) for hazard assessment. Lastly, the `weather-usa` dataset [39] includes daily weather reports for U.S. cities, specifically from January 3, 2016.

Table 3: Summary of Real World Datasets

Dataset	M	l_I	l_S	Description
`covid-kor [37]`	9	16	9	COVID-19 patients (Korea)
`covid-usa [36]`	58	16	22	COVID-19 patients (US)
`gdacs [38]`	9	16	16	Disaster Alert (Global)
`weather-usa [39]`	304	16	128	Weather data (US)

9.2 Core Functionality Optimizations (RQ1)

Innermost Optimization. As shown in Fig. 3a, Fig. 3b, and Fig. 3c, the performance of HomCompLE and HomCompL is similar on both CPU and GPU, with minor variations across bit sizes. In contrast, HomEQ demonstrates a clear advantage on GPU, significantly outperforming CPU in both 16-bit and 32-bit operations. For HomCompLE and HomCompL, the maximum speed-up at $n_p = 8$ is modest (32-bit CPU/GPU: $1.43\times$, 16-bit CPU: $1.41\times$, 16-bit GPU: $1.35\times$). For HomEQ, the speed-up on GPU is more pronounced, reaching $3.20\times$ for 16-bit and $4.17\times$ for 32-bit.

All three functions (HomCompLE, HomCompL, and HomEQ) reach optimal performance at $n_p = 8$, beyond which further increases in parallel processing units yield diminishing returns. This is attributed to overheads such as communication between processing units. While the theoretical complexity suggests a linear increase in speed-up with additional processing units, practical limitations, including synchronization and communication costs, prevent this from being realized. Thus, allocating 8 processing units, with a preference for GPU in the case of HomEQ, provides the best balance between performance and resource efficiency.

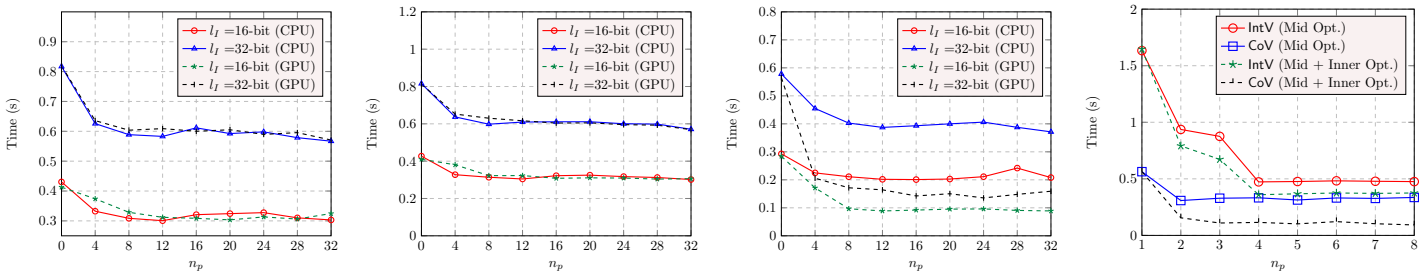
Mid-Level Optimization: Validation Modules. In our experiment, we first applied optimization at the mid-level of the computation hierarchy, focusing on the operational modes IntV and CoV (denoted as IntV (Mid Opt.) and CoV (Mid Opt.) in Fig. 3d). In this setting, parallel processing units were allocated to each validation instance. For IntV, the maximum speed-up was observed at $n_p = 4$, achieving a $3.45\times$ improvement over the non-optimized baseline. We further applied inner-level optimization—targeting homomorphic comparison functions such as HomCompLE—in a GPU-accelerated configuration referred to as IntV (Mid + Inner Opt.) in Fig. 3d. This setting achieved a maximum speed-up of $4.54\times$ using $n_p = 32$ processing units. These results demonstrate that IntV benefits from both mid-level and innermost-level optimizations, though gains diminish beyond $n_p = 4$.

For CoV, the maximum speed-up was obtained at $n_p = 2$, achieving a $1.83\times$ improvement over the non-optimized baseline. Similar to IntV, the GPU-accelerated version—denoted as CoV (Mid + Inner Opt.) in Fig. 3d—yielded a significantly higher speed-up of $6.09\times$. This demonstrates the effectiveness of homomorphic equality operations (e.g., HomEQ) when parallelized on GPU architectures. These results are consistent with our theoretical parallel resource complexity summarized in Table 2, where $d = 4$ for IntV and $d = 2$ for CoV, corresponding to the number of parallel validation steps.

The observed speed-ups suggest that our theoretical model closely aligns with the experimental findings. Based on this evaluation, the optimal number of processing units was determined to be $n_p = 4$ for IntV and $n_p = 2$ for CoV, with additional performance gains enabled by innermost-level optimizations in the GPU-enhanced configurations. A summary of speed-ups across key operations—including HomCompLE, HomCompL, HomEQ, IntV, and CoV—is presented in Fig. 4.

Mid-Level Optimization: HomBitwiseAND. We evaluated the performance of the HomBitwiseAND algorithm by varying the service length (l_S) and the number of parallel processing units (n_p), including GPU acceleration. The results, as shown in Fig. 5a, demonstrate a significant time performance improvement compared to the baseline. Both CPU and GPU optimizations achieve similar performance for smaller service lengths, particularly when $n_p = 16$ or $n_p = 32$, indicating that either approach can be used effectively depending on the available hardware. However, as n_p increases, we observed linear scaling of performance improvements up to the point where $n_p \leq l_S$, beyond which further increases in parallel units provide diminishing returns. This is particularly evident with larger service lengths, where GPU acceleration initially shows superior performance compared to CPU, reaching a maximum speed-up of $11.66\times$ at $l_S = 32$ (see Fig. 6a). Despite this, for even larger service lengths, such as $l_S = 48$ and $l_S = 64$, the benefits of increasing n_p on the GPU begin to decline. This decrease in performance is due to memory bandwidth limitations and data transfer overhead between the CPU and GPU. Thus, while GPU optimization remains preferable for larger workloads, the effectiveness is constrained by memory bottlenecks.

Outermost Optimization: Aggregation via HomSum. We as-



(a) HomCompLE (b) HomCompL (c) HomEQ (d) IntV and CoV

Figure 3: Combined time performance comparison across different algorithms and parallel processing units (n_p) for $\lambda = 128$.

Table 4: Execution time and speed-up across different optimization levels for various private information retrieval strategies using real-world datasets. The experiments evaluate bounding box validation (IntV), coordinate validation (CoV), and identifier matching (IdM) across four datasets.

Dataset	Application Scenario	Method	Baseline (None)	Outermost	Outer + Mid Opt.	All
covid-kor [37]	Pandemic Alert	IntV	16.252	3.304(4.92×)	2.783(5.84×)	2.512 (6.47×)
	City Matching	IdM	4.44331	1.47247(3.02×)	1.21494(3.66×)	1.13875 (3.90×)
covid-usa [36]	Pandemic Alert	IntV	113.44	10.6831(10.62×)	10.1528(11.17×)	9.82627 (11.55×)
	State Matching	IdM	68.4045	8.57437(7.98×)	7.8827(8.68×)	7.95777 (8.59×)
gdacs [38]	Disaster Alert	CoV	7.82023	1.8384(4.25×)	1.70333(4.59×)	1.12841 (6.93×)
weather-usa [39]	Weather Info	IdM	725.38	67.3917(10.76×)	65.5437(11.07×)	65.7115 (11.04×)

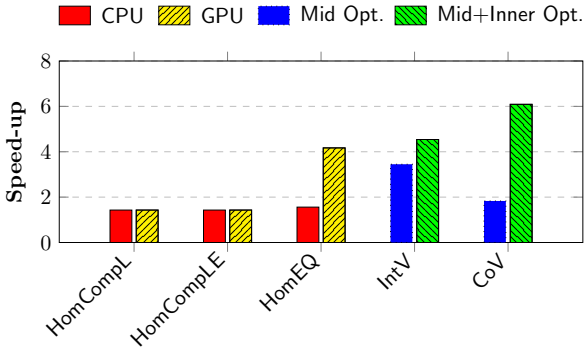
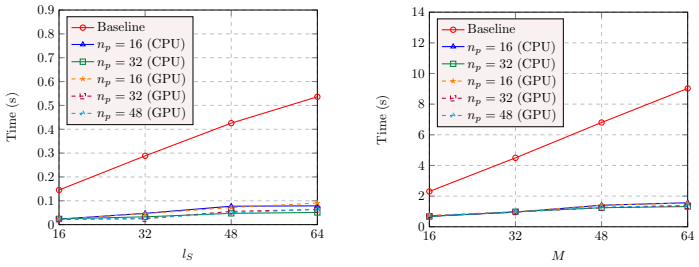


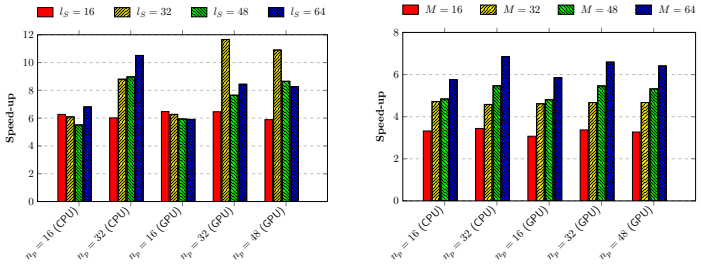
Figure 4: Speed-up summary for mid-level and innermost optimizations across core operations in VeLoPIR.



(a) HomBitwiseAND by l_S (b) HomSum by M

Figure 5: Time performance comparison of HomBitwiseAND and HomSum across different (a) service lengths (l_S) and (b) data sizes (M), with CPU and GPU optimizations.

sessed the performance of the HomSum algorithm by varying the number of data elements (M) and the number of parallel processing units (n_p), including GPU acceleration. As shown in Fig. 5b, both CPU and GPU implementations demonstrate significant performance gains compared to the baseline. Fig. 6b further illustrates the speed-up achieved across different dataset sizes (M). As the number of data elements increases (from $M = 16$ to $M = 64$), the speed-up remains consistent, with up to $6.85\times$ improvement on CPU with $n_p = 32$. This is primarily due to the parallel reduction technique used in HomSum, which reduces the dataset by half with each iteration, enabling efficient scaling with M. The number of processing units (n_p) was sufficient for the dataset sizes tested, avoiding any resource bottlenecks. Additionally, CPU and GPU performance are closely matched across all dataset sizes. For smaller datasets ($M = 16$, $M = 32$), the CPU slightly outperforms the GPU, while for larger datasets ($M = 48$, $M = 64$), the GPU has a marginal edge. However, the differences are minimal, demonstrating that both CPU and GPU offer comparable levels of acceleration for this task.



(a) HomBitwiseAND (b) HomSum

Figure 6: Speed-up comparison of different (a) service lengths (l_S) and (b) data sizes (M) with CPU or GPU optimization.

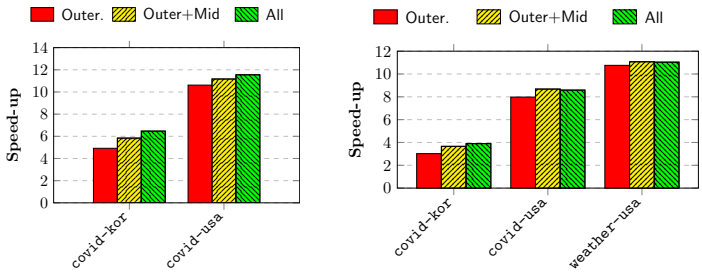
9.3 Performance and Efficiency Analysis (RQ2)

Optimization Strategy. In our evaluation, we implement three levels of optimization to enhance the efficiency of VeLoPIR, corresponding to the system's computational hierarchy (see Table 2 for details). Outermost optimization targets large-scale processing and aggregation steps, parallelizing operations across records using GPU resources. Mid-level optimization focuses on the validation modes IntV and CoV, as well as the HomBitwiseAND operation, leveraging CPU and GPU parallelism where appropriate. Innermost optimization targets fundamental FHE comparison primitives, such as HomCompLE, HomCompL, and HomEQ, which are accelerated using GPU to maximize bit-level parallelism.

VeLoPIR (IntV) Efficiency Compared to Baseline. From Table 4, we observe significant performance improvements achieved by our optimized VeLoPIR over the baseline.

For the `covid-kor` dataset, used in [15], we evaluated VeLoPIR using all three optimization levels under the IntV mode, consistent with the baseline setup. Our optimized version reduced the execution time from 16.252 seconds to 2.512 seconds, resulting in a $6.47\times$ overall speed-up (see Fig. 7a). The most significant contribution came from outermost optimization, yielding a $4.92\times$ speed-up, while mid-level and innermost optimizations provided additional, though smaller, improvements.

For the `covid-usa` dataset, which is significantly larger (with 58 entries compared to 9 in `covid-kor`), outermost optimization again showed the most impact, achieving a speed-up of $10.62\times$. The larger dataset size is well-suited for GPU resource allocation at the outer loop level, enabling efficient parallel reduction across multiple records. However, for this dataset, the gains from mid-level and innermost optimizations diminished, likely due to overhead from CPU-GPU memory transmission as the dataset size and service length increased. This contrast between the `covid-kor` and `covid-usa` datasets highlights the scalability of outermost optimization, while also suggesting potential bottlenecks in deeper optimizations for larger inputs (see the comparison of mid-level and innermost speed-ups between `covid-kor` and `covid-usa` in Fig. 7a).



(a) VeLoPIR using IntV (b) VeLoPIR using IdM

Figure 7: Speed-up comparison across different optimization levels in VeLoPIR.

Efficiency Evaluation of VeLoPIR (IdM). We conducted experiments using the IdM mode across three datasets—`covid-kor`, `covid-usa`, and `weather-usa`—to evaluate the scalability of our optimization strategy with respect to dataset size (M) and service length (l_S). The speed-up results are illustrated in Fig. 7b, with detailed execution times provided in Table 4.

As the dataset size increases, the overall speed-up also improves. For the smaller `covid-kor` dataset, we achieved a speed-up of $3.90\times$, while the `covid-usa` dataset yielded $8.59\times$. The largest dataset, `weather-usa`, achieved the highest speed-up of $11.04\times$, demonstrating the scalability of the IdM mode under our optimization framework.

Notably, the `weather-usa` dataset ($M = 304$, $l_S = 128$) underscores the benefit of large-scale parallelism, especially at the outermost and mid-level optimization levels. However, we observed diminishing returns from innermost optimization as the dataset size increased. For instance, in the `covid-kor` dataset, innermost optimization contributed a modest gain (from $3.66\times$ to $3.90\times$), whereas for `covid-usa` and `weather-usa`, the speed-up slightly declined (from $8.68\times$ to $8.59\times$ and from $11.07\times$ to $11.04\times$, respectively). This suggests that GPU-accelerated innermost operations can become bottlenecked by memory bandwidth limitations at scale. Addressing this issue would require improved hardware support for more efficient data transfer between CPU and GPU.

Efficiency Evaluation of VeLoPIR (CoV). We conducted experiments using the CoV mode with the `gdacs` dataset ($M = 9$, $l_S = 16$). A total speed-up of $6.93\times$ was achieved, primarily driven by outermost optimization, which contributed $4.25\times$. Additional gains were obtained through mid-level and innermost optimizations, with innermost optimization alone yielding a $2.34\times$ improvement via parallelized HomEQ operations. Notably, no significant memory bandwidth issues or diminishing returns were observed in this setting, likely due to the relatively small dataset size and balanced CPU-GPU data transfer overhead.

Scalability in Practical Settings. While VeLoPIR's design allows for high degrees of parallelism, its real—world scalability is affected by memory bandwidth constraints—particularly between CPU and GPU. As observed in Fig. 5a, the performance of HomBitwiseAND, a mid-level optimization component, begins to saturate as the number of parallel units n_p increases under a fixed service length l_S. A similar saturation trend is seen in Fig. 5b for HomSum, which belongs to the outermost optimization layer, as the dataset size M is held constant. These effects result in a divergence between theoretical and empirical speed-up, as summarized in Table 4.

In contrast, more lightweight primitive operations such as HomCompLE and HomEQ, shown in Fig. 3, continue to follow expected scaling behavior, particularly for modest parallelism levels ($n_p = 4$ or 8). This is because these primitives are less dependent on high-throughput memory access and thus less affected by interconnect bottlenecks. These results suggest that for large-scale deployments of VeLoPIR, improvements in memory bandwidth—especially between host and device—would allow the system to more closely approach its theoretical parallel performance bounds.

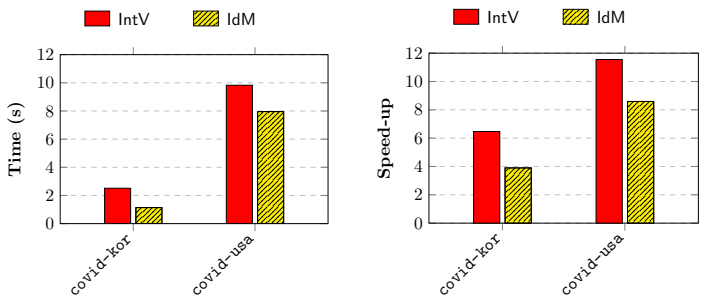
9.4 Operational Mode Applicability (RQ3)

Information Alerts. In scenarios such as pandemic alerts (`covid-kor` and `covid-usa` datasets), both IntV and IdM modes can be applied. However, IdM is generally more efficient,

as it directly compares encrypted locational identifiers (e.g., city or state names) using homomorphic equality (HomEQ), thereby avoiding bounding box computations and reducing circuit depth.

This performance advantage is evident in our results (see Fig. 8a and Fig. 8b). For the `covid-kor` dataset, the baseline execution time for IntV was 16.252 seconds, which our optimized version reduced to 2.512 seconds—a $6.47\times$ speed-up. In contrast, IdM reduced the baseline of 4.443 seconds to 1.138 seconds, yielding a $3.90\times$ improvement.

The scalability of these modes is further highlighted by the results on the `covid-usa` dataset. Given its larger size, IntV achieved an $11.55\times$ speed-up (from 113.44 to 9.826 seconds), while IdM reduced the baseline from 68.404 to 7.957 seconds, resulting in an $8.59\times$ improvement.



(a) Execution Time (IntV vs. IdM) (b) Speed-up (IntV vs. IdM)

Figure 8: Time performance and speed-up comparison for IntV and IdM modes using the `covid-kor` and `covid-usa` datasets (information alert scenario).

Emergency Alerts. The CoV mode is well-suited for emergency alert scenarios, as demonstrated by our experiments using the real-world `gdacs` dataset (Global Disaster Alert and Coordination System). Although relatively small in size ($M = 9, l_S = 16$), the dataset represents realistic use cases such as satellite-based disaster monitoring. Its structure aligns well with the targeted nature of CoV, which enables exact coordinate-based validation for timely alerts.

The speed-up results indicate that CoV is highly efficient, reducing retrieval time to just 1.1284 seconds—a $6.93\times$ improvement over the baseline. This demonstrates the effectiveness of CoV in rapidly and accurately delivering alerts for specific geographic locations in disaster scenarios, without the overhead associated with more general-purpose modes like IntV or IdM.

Moreover, the CoV mode is broadly adaptable to other emergency use cases, such as crime alerts or evacuation warnings, where users in predefined coordinates require precise and immediate notifications.

10 Conclusion

We introduced VeLoPIR, a fast and versatile location-based PIR system that protects user locational privacy in real-world information and emergency alert scenarios. Through a set of flexible operational modes, VeLoPIR efficiently handles diverse datasets while preserving privacy. Performance evaluations on real-world datasets demonstrated significant speed-ups through parallel processing on both CPU and GPU, confirming the system's scalability. We also provided formal correctness proofs along with comprehensive security and privacy analyses, validating VeLoPIR's effectiveness and robustness in practical deployments.

References

[1] S. Thompson and C. Warzel, "Your apps know where you were last night, and they're not keeping it secret," *The New York Times*, Dec. 10, 2018. [Online]. Available: `https://www.nytimes.com/interactive/2018/12/10/business/location-data-privacy-apps.html`

[2] Z. Whittaker, "A breach of Gravy Analytics' huge trove of location data threatens the privacy of millions," *TechCrunch*, Jan. 13, 2025. [Online]. Available: `https://techcrunch.com/2025/01/13/gravy-analytics-data-broker-breach-trove-of-location-data-threatens-privacy-millions/`

[3] "Google will pay Texas $1.4B to settle claims the company collected users' data without permission," *AP News*, May 9, 2025. [Online]. Available: `https://apnews.com/article/8097e181cc7cb8522781db8a9a897eea`

[4] W. Diffie and M. E. Hellman, "New directions in cryptography," in *Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman*, pp. 365–390, 2022.

[5] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, Boca Raton, FL, USA: CRC Press, 2018.

[6] R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On data banks and privacy homomorphisms," *Found. Secure Comput.*, vol. 4, no. 11, pp. 169–180, 1978.

[7] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, "Private information retrieval," *J. ACM*, vol. 45, no. 6, pp. 965–981, 1998.

[8] A. Hamlin, R. Ostrovsky, M. Weiss, and D. Wichs, "Private anonymous data access," in *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Darmstadt, Germany, May 19–23, 2019, pp. 244–273.

[9] P. Ananth, K.-M. Chung, X. Fan, and L. Qian, "Collusion-resistant functional encryption for RAMs," in *Int. Conf. Theory and Application of Cryptology and Information Security*, 2022, pp. 160–194.

[10] A. Beimel, Y. Ishai, and T. Malkin, "Reducing the servers' computation in private information retrieval: PIR with preprocessing," in *Advances in Cryptology–CRYPTO 2000: 20th Annual Int. Cryptol. Conf.*, Santa Barbara, CA, USA, Aug. 20–24, 2000, pp. 55–73.

[11] E. Boyle, Y. Ishai, R. Pass, and M. Wootters, "Can we access a database both locally and privately?," in *Theory of Cryptography: 15th Int. Conf., TCC 2017*, Baltimore, MD, USA, Nov. 12–15, 2017, pp. 662–693.

[12] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," in *Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security*, Hanoi, Vietnam, Dec. 4–8, 2016, pp. 3–33.

[13] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast fully homomorphic encryption over the torus," *J. Cryptol.*, vol. 33, no. 1, pp. 34–91, 2020.

[14] M. Jain, P. Singh, and B. Raman, "SHELBRS: Location-based recommendation services using switchable homomorphic encryption," in *Int. Conf. Security, Privacy, and Appl. Cryptogr. Eng.*, 2021, pp. 63–80.

[15] J. S. Yoo, M. Y. Hong, J. W. Heo, K. H. Lee, and J. W. Yoon, "Fast private location-based information retrieval over the torus," in *Proc. 2024 IEEE Int. Conf. Adv. Video Signal Based Surveillance (AVSS)*, 2024, pp. 1–7.

[16] H. Corrigan-Gibbs, A. Henzinger, and D. Kogan, "Single-server private information retrieval with sublinear amortized time," in *Advances in Cryptology–EUROCRYPT 2022: 41st Annual Int. Conf. Theory and Applications of Cryptographic Techniques*, 2022, pp. 3–33.

[17] W.-K. Lin, E. Mook, and D. Wichs, "Doubly efficient private information retrieval and fully homomorphic RAM computation from ring LWE," in *Proc. 55th Annu. ACM Symp. Theory of Comput. (STOC 2023)*, pp. 595–608, 2023.

[18] Z. Brakerski and V. Vaikuntanathan, "Fully homomorphic encryption from ring-LWE and security for key dependent messages," in *Advances in Cryptology–CRYPTO 2011: 31st Annu. Int. Cryptol. Conf.*, Santa Barbara, CA, USA, Aug. 14–18, 2011, pp. 505–524.

[19] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Int. Conf. Theory and Appl. Cryptol. Techniques*, 1999, pp. 223–238.

[20] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Inf. Theory*, vol. 31, no. 4, pp. 469–472, 1985.

[21] Y. An, S. Lee, S. Jung, H. Park, Y. Song, and T. Ko, "Privacy-oriented technique for COVID-19 contact tracing (PROTECT) using homomorphic encryption: Design and development study," *J. Med. Internet Res.*, vol. 23, no. 7, pp. e26371, 2021.

[22] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *Annu. Cryptol. Conf.*, 2012, pp. 868–886.

[23] J. Hoffstein, J. Pipher, and J. H. Silverman, "NTRU: A ring-based public key cryptosystem," in *Algorithmic Number Theory (ANTS III)*, 1998, pp. 267–288.

[24] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *J. ACM*, vol. 56, no. 6, pp. 1–40, 2009.

[25] D. Stehlé, R. Steinfeld, K. Tanaka, and K. Xagawa, "Efficient public key encryption based on ideal lattices," in *Int. Conf. Theory Appl. Cryptol. Inf. Security*, 2009, pp. 617–635.

[26] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st Annu. ACM Symp. Theory Comput.*, 2009, pp. 169–178.

[27] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology–ASIACRYPT 2017*, Hong Kong, Dec. 3–7, 2017, pp. 409–437.

[28] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," *ACM Trans. Comput. Theory*, vol. 6, no. 3, pp. 1–36, 2014.

[29] N. O. Tippenhauer, C. Pöpper, K. B. Rasmussen, and S. Capkun, "On the requirements for successful GPS spoofing attacks," in *Proc. 18th ACM Conf. Comput. Commun. Security*, 2011, pp. 75–86.

[30] K. C. Zeng, Y. Shu, S. Liu, Y. Dou, and Y. Yang, "A practical GPS location spoofing attack in road navigation scenario," in *Proc. 18th Int. Workshop Mobile Comput. Syst. Appl.*, 2017, pp. 85–90.

[31] Z. M. Kassas, J. Khalife, A. A. Abdallah, and C. Lee, "I am not afraid of the GPS jammer: Resilient navigation via signals of opportunity in GPS-denied environments," *IEEE Aerosp. Electron. Syst. Mag.*, vol. 37, no. 7, pp. 4–19, 2022.

[32] A. Grant, P. Williams, N. Ward, and S. Basker, "GPS jamming and the impact on maritime navigation," *J. Navig.*, vol. 62, no. 2, pp. 173–187, 2009.

[33] E. Zeydan, Y. Turk, B. Aksoy, and S. B. Ozturk, "Recent advances in post-quantum cryptography for networks: A survey," in *2022 7th Int. Conf. Mobile and Secure Services (MobiSecServ)*, 2022, pp. 1–8.

[34] S. Paul, P. Scheible, and F. Wiemer, "Towards post-quantum security for cyber-physical systems: Integrating PQC into industrial M2M communication," *J. Comput. Security*, vol. 30, no. 4, pp. 623–653, 2022.

[35] Z. Liu, K.-K. R. Choo, and J. Grossschädl, "Securing edge devices in the post-quantum Internet of Things using lattice-based cryptography," *IEEE Commun. Mag.*, vol. 56, no. 2, pp. 158–162, 2018.

[36] E. Dong, H. Du, and L. Gardner, "An interactive web-based dashboard to track COVID-19 in real time," *Lancet Infect. Dis.*, vol. 20, no. 5, pp. 533–534, 2020, doi: 10.1016/S1473-3099(20)30120-1. [Online]. Available: https://github.com/CSSEGISandData/COVID-19.

[37] Korea Disease Control and Prevention Agency (KDCA), "COVID-19 daily incidences in nine major cities in Korea," 2021. [Online]. Available: https://ncv.kdca.go.kr/pot/cv/trend/dmstc/selectMntrgSttus.do.

[38] Global Disaster Alert and Coordination System (GDACS), "Global Disaster Alert and Coordination System (GDACS) alerts and impact estimations," 2024. [Online]. Available: https://www.gdacs.org/Alerts/default.aspx.

[39] National Oceanic and Atmospheric Administration (NOAA), "Daily weather reports for U.S. cities," 2016. [Online]. Available: https://www.noaa.gov/climate.

A CORRECTNESS PROOF

A.1 Proof of Theorem 1

Proof. The main part of the HomCompLE algorithm relies on the combination of homomorphic gates, specifically the HomXNOR gate followed by the HomMUX gate, to evaluate the encrypted inputs.

First, the algorithm initializes t_2 with a trivial TLWE ciphertext $\mathsf{Enc}_{\mathbf{s}}^{\mathsf{TLWE}}(\mathbf{0}, \frac{1}{8})$, which is an encryption of 1 under the secret key $\mathbf{s}$. This assumes $\mathsf{ct}_1 \leq \mathsf{ct}_2$ initially.

If ct_1 and ct_2 are equal, then for all bits, the HomXNOR gate will output $t_1 = \mathsf{Enc}_{\mathbf{s}}(1)$, preserving $t_2 = \mathsf{Enc}_{\mathbf{s}}(1)$ throughout the loop. The final output, r, becomes $\mathsf{Enc}_{\mathbf{s}}(1)$, confirming $\mathsf{ct}_1 \leq \mathsf{ct}_2$ as expected.

Consider the highest bit j where $\mathsf{ct}_1[j]$ and $\mathsf{ct}_2[j]$ differ:

$$\text{(Case 1): } \mathsf{ct}_1[j] = 1, \mathsf{ct}_2[j] = 0$$
$$\text{(Case 2): } \mathsf{ct}_1[j] = 0, \mathsf{ct}_2[j] = 1$$

- Case 1. Here, HomXNOR outputs $t_1 = \mathsf{Enc}_{\mathbf{s}}(0)$. Consequently, HomMUX selects $\mathsf{ct}_2[j]$, setting $t_2 = \mathsf{Enc}_{\mathbf{s}}(0)$. Thus, the result is $\mathsf{Enc}_{\mathbf{s}}(0)$, correctly indicating that $\mathsf{ct}_1 > \mathsf{ct}_2$.
- Case 2. In this scenario, HomXNOR outputs $t_1 = \mathsf{Enc}_{\mathbf{s}}(0)$, leading HomMUX to select $\mathsf{ct}_2[j]$, thus $t_2 = \mathsf{Enc}_{\mathbf{s}}(1)$. Therefore, the result is $\mathsf{Enc}_{\mathbf{s}}(1)$, correctly showing that $\mathsf{ct}_1 < \mathsf{ct}_2$.

The HomMUX at the end considers t_0, which indicates differing signs between ct_1 and ct_2. If signs differ, t_0 directs the selection to $\mathsf{ct}_1[l_I - 1]$. This ensures correct handling of signed numbers:

- If ct_1 is negative and ct_2 is positive, $\mathsf{ct}_1 < \mathsf{ct}_2$ holds, setting the result as $\mathsf{Enc}_{\mathbf{s}}(1)$.
- Conversely, if ct_1 is positive and ct_2 is negative, $\mathsf{ct}_1 > \mathsf{ct}_2$, leading to $\mathsf{Enc}_{\mathbf{s}}(0)$.

□

A.2 Proof of Corollary 1

Proof. The algorithm $\mathsf{HomCompL}(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{evk})$ builds upon the logic established in Theorem 1 for the HomCompLE algorithm. The key difference in HomCompL lies in the initialization of the variable t_2, which is set to $\mathsf{Enc}_{\mathbf{s}}^{\mathsf{TLWE}}(\mathbf{0}, -\frac{1}{8})$, corresponding to an encryption of 0.

In the HomCompLE algorithm, t_2 was initialized to an encryption of 1, representing the assumption that $\mathsf{ct}_1 = \mathsf{ct}_2$. Here, by initializing t_2 to 0, we instead assume that $\mathsf{ct}_1 \neq \mathsf{ct}_2$. The iterative loop in HomCompL uses the HomXNOR and HomMUX gates to compare the bits of ct_1 and ct_2. If all bits of ct_1 and ct_2 are equal, t_2 remains as 0, which is the correct result since $\mathsf{ct}_1 = \mathsf{ct}_2$ means $\mathsf{ct}_1 < \mathsf{ct}_2$ is false.

The variable t_0, as in HomCompLE, accounts for cases where ct_1 and ct_2 have different signs. If the signs differ, t_0 directs the output based on the sign of ct_1. This ensures that the algorithm correctly handles all cases where $\mathsf{ct}_1 < \mathsf{ct}_2$, including when the signs are different.

Thus, by initializing t_2 to 0, HomCompL ensures that when $\mathsf{ct}_1 = \mathsf{ct}_2$, the output is $\mathsf{Enc}_{\mathbf{s}}(0)$, and when $\mathsf{ct}_1 < \mathsf{ct}_2$, the output is $\mathsf{Enc}_{\mathbf{s}}(1)$. This proves that the algorithm correctly evaluates $\mathsf{ct}_1 < \mathsf{ct}_2$ as stated. □

A.3 Proof of Lemma 1

Proof. The algorithm starts by initializing the result r as a trivial TLWE encryption of 1, denoted as $\mathsf{Enc}_{\mathbf{s}}^{\mathsf{TLWE}}(0, \frac{1}{8})$ assuming that $\mathsf{ct}_1 = \mathsf{ct}_2$ by default.

In each iteration of the loop, the HomXNOR gate compares corresponding bits of ct_1 and ct_2. If the bits are equal, HomXNOR outputs $\mathsf{Enc}_{\mathbf{s}}(1)$; otherwise, it outputs $\mathsf{Enc}_{\mathbf{s}}(0)$. The result t from each bit comparison is then combined with the current value of r using the HomAND operation.

The HomAND operation is crucial: it ensures that if any bit of ct_1 and ct_2 differs, the final result r becomes $\mathsf{Enc}_{\mathbf{s}}(0)$, indicating inequality. Conversely, if all corresponding bits of ct_1 and ct_2 are identical, the final value of r remains $\mathsf{Enc}_{\mathbf{s}}(1)$, confirming that ct_1 and ct_2 encrypt the same value.

Thus, the algorithm correctly evaluates the equality of z_1 and z_2 under homomorphic encryption, as claimed. □

A.4 Proof of Theorem 2

Proof. The correctness of this algorithm can be verified by analyzing the behavior of the homomorphic comparison operations (HomCompL and HomCompLE operations).

For the latitude, $v_{x_{left}}$ and $v_{x_{right}}$ will both be $\mathsf{Enc}_{\mathbf{s}}(1)$ only when $x_{left} \leq x < x_{right}$. Thus, the combined result $v_x = \mathsf{HomAND}(v_{x_{left}}, v_{x_{right}}, \mathsf{evk})$ will be $\mathsf{Enc}_{\mathbf{s}}(1)$ only if this condition holds. Conversely, consider a case where $x < x_{left}$; in this scenario, $v_{x_{left}}$ will be $\mathsf{Enc}_{\mathbf{s}}(0)$, ensuring that v_x becomes $\mathsf{Enc}_{\mathbf{s}}(0)$ irrespective of the value of $v_{x_{right}}$.

Similarly, the combined result $v_y = \mathsf{HomAND}(v_{y_{left}}, v_{y_{right}}, \mathsf{evk})$ will be $\mathsf{Enc}_{\mathbf{s}}(1)$ only if $y_{left} \leq y < y_{right}$ holds.

The final validation $v = \mathsf{HomAND}(v_x, v_y, \mathsf{evk})$ will output $\mathsf{Enc}_{\mathbf{s}}(1)$ if both latitude and longitude conditions are satisfied, en-

suring the coordinate (x, y) is within the interval. Otherwise, it outputs $\mathsf{Enc}_{\mathbf{s}}(0)$. □

B SECURITY PROOF

B.1 Proof of Corollary 2

Proof. The server's goal is to retrieve the secret key $\mathbf{s}$ from the encrypted dataset $\mathsf{Enc}_{\mathsf{pk}}(\mathbf{D})$. This problem is essentially identical to the one discussed in Theorem 6, since the ciphertext ct in $\mathsf{Enc}_{\mathsf{pk}}(\mathbf{D})$ is of the form $\mathsf{ct} \leftarrow (\mathbf{0}, \mathsf{Ecd}(x)) + \mathsf{pk}$. Thus, the server must extract the secret key from pk, which consists of up to $W = Ml_Id + Ml_S$ TLWE samples. Therefore, the server's advantage is bounded by:

$$\mathrm{Adv}^{\mathsf{SE}}_{\mathrm{TLWE}_{1\text{ to }W}} < \frac{O(W)}{2^{\lambda}}$$

where $W = Ml_Id + Ml_S$. □

C ADDITIONAL SUPPORTING INFORMATION

In this section, we briefly explain the bootstrapping procedure in TFHE, which is critical for performing homomorphic gate operations. We also provide the TFHE parameters used during the evaluation of VeLoPIR, followed by a summary of the notations used throughout the paper.

C.1 TFHE Parameters

Table 5 lists the cryptographic parameters for the TFHE scheme at the 128-bit security level, as used in [13].

Table 5: Cryptographic parameters for TFHE 128-bit security level

Parameter		Value
TLWE Dimension	n	630
TRLWE Dimension	k	1
Polynomial Size	N	1024
Key Switch Base Log	$\log_2(\beta_{KS})$	2
Key Switch Level	ℓ_{KS}	8
Key Switch Standard Deviation	σ_{KS}	2^{-15}
Bootstrapping Key Base Log	$\log_2(\beta_{BK})$	7
Bootstrapping Key Level	ℓ_{BK}	3
Bootstrapping Key Standard Deviation	σ_{BK}	2^{-25}