QPADL: Post-Quantum Private Spectrum Access with Verified Location and DoS Resilience

Saleh Darzi, Saif Eddine Nouma, Kiarash Sedghighadikolaei, Attila Altay Yavuz *Senior IEEE,*
salehdarzi@usf.edu, saifeddinenouma@usf.edu, kiarashs@usf.edu, attilaayavuz@usf.edu

***Abstract*—With advances in wireless communication and growing spectrum scarcity, Spectrum Access Systems (SASs) offer an opportunistic solution but face significant security challenges. Regulations require disclosure of location coordinates and transmission details, exposing user privacy and anonymity during spectrum queries, while the database operations themselves permit Denial-of-Service (DoS) attacks. As location-based services, SAS is also vulnerable to compromised or malicious users conducting spoofing attacks. These threats are further amplified given the quantum computing advancements. Thus, we propose `QPADL`, the first post-quantum (PQ) secure framework that simultaneously ensures privacy, anonymity, location verification, and DoS resilience while maintaining efficiency for large-scale spectrum access systems. `QPADL` introduces SAS-tailored private information retrieval for location privacy, a PQ-variant of Tor for anonymity, and employs advanced signature constructions for location verification alongside client puzzle protocols and rate-limiting technique for DoS defense. We formally assess its security and conduct a comprehensive performance evaluation, incorporating GPU parallelization and optimization strategies to demonstrate practicality and scalability.**

***Index Terms*—Privacy and Anonymity, Post-Quantum Security, Location Proof, Spectrum Access Systems, Counter DoS.**

I. Introduction

The rapid growth of wireless technologies (e.g., mobile and IoT) combined with regulated spectrum allocation (e.g., FCC in the U.S. [1]) has led to spectrum scarcity. Cognitive Radio Networks (CRNs) mitigate this challenge by enabling Secondary Users (SUs) to opportunistically access unused licensed channels [2]. At the core of this model lies the Spectrum Access System (SAS), which dynamically allocates spectrum through geo-location databases (DBs) [3]. Despite its effectiveness, SAS poses major privacy and security risks: continuous reporting of locations and transmission details compromises user privacy and anonymity [4], while its location-centric design enables spoofing, falsified data, and unauthorized access [5]. The broadcast nature of spectrum communication, reliance on databases, and widespread use of low-cost devices further expose SAS to denial-of-service (DoS) attacks [6]. These challenges are intensified by quantum computing, which threatens classical cryptographic protections [7]. *While prior efforts to address these security challenges in spectrum access remain isolated, no existing work offers a unified PQ-secure framework that simultaneously ensures location privacy, anonymity, verifiable location, and DoS resilience under FCC-compliant SAS constraints.*

Saleh Darzi, Saif Eddine Nouma, Kiarash Sedghighadikolaei, and Attila A. Yavuz are with the the Bellini College of Artificial Intelligence, Cybersecurity, and Computing at the University of South Florida (USF), Tampa, Fl, 33620.

A. Related Works and Open Problems

(i) Location Privacy and Anonymity in SAS: Centralized SAS, mandated by the FCC, operates through multiple geolocation databases to facilitate dynamic spectrum sharing among governmental, commercial, licensed and unlicensed users [1]. Access requires disclosure of sensitive data such as exact geographic coordinates, channel preferences, usage patterns, and transmission parameters, leaving users vulnerable to identity tracing, behavioral profiling, and lifestyle inference attacks [4], [8]. However, existing approaches remain inadequate: k-anonymity and pseudo-identifier methods [9] lack provable security and provide only weak guarantees unless large anonymity sets are used, which is infeasible in large-scale SAS deployments; differential privacy-based techniques [10], while theoretically sound, significantly degrade the accuracy of spectrum availability information; and Private Information Retrieval (PIR)-based schemes [2], [5] focus solely on location privacy while neglecting anonymity, assuming honest uncompromised users, and ignoring location spoofing. These gaps highlight the necessity for efficient, provably secure mechanisms that provide strong anonymity and location privacy under realistic network assumptions, without sacrificing performance or user experience [6].

(ii) Location Verification and Spoofing Attack Resistance: SAS, as a location-based platform, relies on accurate user location data for fair and efficient spectrum allocation, making it vulnerable to adversaries who impersonate users or submit falsified locations, leading to interference, disruptions, and economic loss [11]. Existing location verification schemes often assume trusted infrastructure, honest participants, or dedicated location servers, assumptions that are unrealistic in rural or infrastructure-limited settings [5]. Furthermore, most approaches fail to preserve location privacy and anonymity from the verifying entities themselves. These limitations underscore the need for a practical and privacy-preserving location verification framework capable of operating under real-world constraints while resisting diverse location-based attacks [12].

(iii) Counter-DoS and Spectrum Management for Next-Gen Network Systems: The widespread deployment of low-cost IoT devices, together with SAS's dependence on geolocation databases, has considerably heightened susceptibility to DoS attacks [13]. These attacks flood the system with illegitimate requests, hindering spectrum coordination and impairing overall system responsiveness, especially during spectrum access and service interactions. Various mitigation strategies have been proposed, including intrusion detection systems (IDSs), blockchain-based access control, and cryptographic methods such as Client Puzzle Protocols (CPPs) [14]. Machine learning

has further advanced IDSs, enabling detection of abnormal traffic patterns with reported accuracies above 95% [3]. However, these methods focus on detection rather than prevention, and their effectiveness depends on continuous access to sensitive user traffic and often private network topology, requirements that are impractical for real-time SAS defense. Also, AI-driven defenses remain susceptible to adversarial manipulation, where attackers exploit model weaknesses to evade detection, potentially incurring significant operational costs [15]. CPPs mitigate malicious requests by requiring clients to solve puzzles before service [16], but at scale they suffer from bottlenecks in puzzle generation, distribution, and parallelization, imposing heavy costs on servers and legitimate users [17]. Outsourcing puzzle generation, as in [6], reduces server load but shifts DoS risks to spectrum databases, which attackers can still overwhelm with queries. Thus, a lightweight rate-limiting mechanism embedded in the spectrum query phase is crucial to maintain availability, counter large-scale DoS, and improve SAS resilience without excessive overhead.

(iv) Spectrum Access in the Post-Quantum Era: The rise of quantum computing threatens the long-term security of wireless networks by breaking classical cryptographic primitives that protect SAS [7]. Protocols that preserve privacy, anonymity, and location verification rely on hardness assumptions vulnerable to quantum attacks, leaving systems exposed to threats such as location disclosure, spoofing, and large-scale DoS [6]. Ensuring durable protection requires adopting Post-Quantum Cryptography (PQC) to preserve privacy, anonymity, and availability against quantum-capable adversaries.

B. Our Contributions

To the best of our knowledge, `QPADL` is the first PQ secure framework for spectrum access that simultaneously achieves several often conflicting objectives, ensuring SUs privacy and anonymity while complying with FCC's strict regulations, even under realistic network settings where malicious or compromised users may attempt location spoofing to gain spectrum advantages or launch DoS attacks to disrupt legitimate access. `QPADL` provides the following desirable properties:

- ***Location Privacy-Preserving and Anonymous Spectrum Access:*** `QPADL` achieves PQ-secure location privacy and anonymity of SUs in spectrum access while abiding by the FCC's regulations. We propose an NIST-compliant PQ-variant of The Onion Routing (Tor) network to anonymize spectrum queries to the Private Spectrum Databases (PSDs) and design three `QPADL` instantiations: `QPADL-ENS` for efficiency, `QPADL-FTR` for robustness, and `QPADL-OOP` for reduced communication. These instantiations are built on various information-theoretic (IT) or PQ multi-server PIR techniques and are tailored for FCC-compliant SAS.
- ***Location Verification and Spoofing Resistance:*** `QPADL` operates under realistic network assumptions with malicious or compromised SUs, leveraging already existing WiFi Access Points (APs) and cellular towers during spectrum query phase to verify SU locations and issue time-sensitive, quantum-safe proofs using signal strength for proximity checks. To the best of our knowledge, this work is the first to employ event-oriented linkable ring signatures (LRSs) for location proofs in wireless networks, eliminating the need for trusted or dedicated infrastructure, while the event-ID's linkability enables limiting the number of proofs acquired. `QPADL` protects SAS against spoofing attacks such as mafia fraud and distance hijacking [11], while preserving the anonymity of both SUs and APs against SAS and PSD servers.
- ***Outsourced DoS Mitigation with SAS Architecture Compliance:*** To overcome the limitations of existing DoS defenses and the added risks of quantum-capable adversaries, `QPADL` introduces an outsourced counter-DoS service built on diverse CPP protocols with PQ-secure puzzle generation (e.g., hash-based, lattice-based) handled by the already existing entities i.e., PSDs. This design mitigates malicious SU and external adversary attacks during service requests and SAS server communications. While outsourcing shifts the DoS target to PSDs, `QPADL` is the first to reinforce their protection through a rate-limiting mechanism based on the linkability of the LRS, ensuring comprehensive DoS resistance without violating SAS architecture or FCC regulations.
- ***Enabling Scalability Through Parallelization and Optimization:*** We conducted comprehensive analytical and empirical evaluations of all `QPADL` instantiations, demonstrating their efficiency and scalability. By leveraging GPU parallelization, acceleration methods, and database compression, we mitigated PIR bottlenecks and reduced PSD-side costs affected by database size. With one to 2^{10} users, GPU acceleration in `QPADL-ENS` yields an average $2.77\times$ speedup on the PSD side and $4.18\times$ in end-to-end delay; `QPADL-FTR` achieves $4.88\times$ and $11.49\times$, respectively; and `QPADL-OOP` delivers $1.71\times$ and $1.68\times$. These results confirm `QPADL`'s strong scalability, particularly beyond 2^{10} SUs per time window.

C. Improvements Over the MILCOM 2024 Conference Paper

This work is an extended version of *IEEE MILCOM-2024 Conference* [6], with 100% more new material and the following significant improvements and additional features:
(i) ***Location Verification Mechanism:*** We propose a PQ-secure location verification mechanism that, under realistic networks with malicious or compromised SUs, leverages LRSs and nearby WiFi APs to resist location spoofing attacks.
(ii) ***Comprehensive DoS Countermeasures:*** We present two `QPADL` instantiations using distinct CPPs and a rate-limiting technique based on LRS linkability to enhance DoS resistance against `PSDs`. The first, `QPADL-HCT`, employs a hash-cash tree for tunable puzzle difficulty with parallelization resistance, while the second, `QPADL-LBP`, integrates lattice-based PoWs to overcome hash-function vulnerabilities to Grover's algorithm, offering a quantum security advantage for counter-DoS.
(iii) ***Various PIR Instantiations:*** Given PIR's critical role in our framework, we design three `QPADL` variants: `QPADL-ENS` for efficiency, `QPADL-FTR` for fault-tolerant robustness, and `QPADL-OOP` for scalability with lower communication costs.
(iv) ***Hardware Acceleration and Optimizations*** We addressed the main bottleneck of [6] (i.e., PSD-side computations), by applying GPU parallelization and acceleration strategies to reduce overhead and improve scalability in large-scale SAS. We also defined the DB structure, detailed setup and content subroutines, and proposed compression techniques.

II. Preliminaries and Building Blocks

Notations: $||$, $|x|$, and $\{0,1\}^k$ denote concatenation, the bit length of a variable, and a k-bit binary value, respectively. $\mathbb{F}$, $GF(2)$, and $\mathbb{Z}$ represent a finite field, the Galois Field of order

2, and the set of integers, respectively. $\{x_i\}_{i=1}^{\ell}$ denotes the tuple $(x_1, x_2, \ldots, x_\ell)$. The notation $x \xleftarrow{\$} \mathcal{S}$ indicates uniform random selection from the set $\mathcal{S}$, and $\mathbf{v}$ denotes a vector. $||.||$ represents the Euclidean norm of a lattice vector. λ is the security parameter, and p is a large prime. Γ is the Gamma function in the lattice distribution, while Λ denotes the lattice with dimension n_Λ. The functions H and H' are cryptographically secure hash functions. $Enc_{sk}(m)$ denotes encryption of a message m using the secret key sk, while sk and pk denote the secret and public keys, respectively. σ, $ctxt$, and $\mathtt{ID}$ refer to the signature, ciphertext, and user identity, respectively.

A. System Architecture

Our system model consists of the following main entities:

- **Federal Communications Commission:** The FCC represents the primary regulatory body that governs the SAS, sets system requirements, and enforces spectrum compliance.
- **Servers:** SAS operators and network service providers (e.g., CRN, web, cloud) that manage spectrum access requests, monitor usage, resolve interference, and related services.
- **Private Spectrum Databases (PSDs):** hird-party managed geolocation DBs (e.g., Google, Spectrum Bridge) that store spectrum availability data [8] and synchronize regularly to maintain consistency under FCC regulations [2].
- **Clients:** Users with wireless-enabled devices (e.g., mobile phones, laptops) who seek to access network services or act as SUs in SASs by requesting frequency data to opportunistically use licensed spectrum without interfering with PUs.
- **Access Points (APs):** $\mathtt{AP}$ denotes the cellular network towers and WiFi access points in the region that form a group and are equipped with synchronized clocks [8].

B. Cryptographic Building Blocks

1) Private Information Retrieval: PIR enables a client to retrieve a data block from a database $\mathtt{DB}$ without revealing its index [18]. In our architecture with synchronized spectrum DBs, we employ a multi-server PIR model where non-colluding servers each hold a copy of the DB [19].

Definition 1. *A multi-server PIR scheme involves three algorithms executed between a client and n non-colluding servers:*

- $\{q_i\}_{i=1}^{n} \leftarrow \mathtt{Client.Query}(\theta, n)$*: The client generates n queries for the target index θ, sending each q_i to server $\mathtt{DB}_i$.*
- $\rho_i \leftarrow \mathtt{DB.Query.Response}(q_i)$*: Each server processes its query and returns a response ρ_i derived from its local $\mathtt{DB}$.*
- $D_\theta \leftarrow \mathtt{Client.BlockReconst}(\{\rho_i\}_{i=1}^{\ell})$*: The client reconstructs the desired D_θ using the ℓ collected responses.*

2) Proof of Work: A PoW mechanism is a cryptographic primitive based on puzzles that are computationally easy to generate but hard to solve, with tunable difficulty level [20].

Definition 2. *A PoW scheme operate as follows:*

- $\Pi \leftarrow \mathtt{Puzzle.Gen}(1^\lambda, \kappa)$*: Given security parameter λ and difficulty κ, the algorithm generates puzzle instance Π.*
- $\Psi \leftarrow \mathtt{PoW}(\Pi, \kappa)$*: Given a puzzle Π and difficulty κ, this algorithm computes a valid solution Ψ.*
- $\{0,1\} \leftarrow \mathtt{PoW.Verify}(\Pi, \Psi)$*: The algorithm returns 1 if Ψ is a valid solution to puzzle Π, and 0 otherwise.*

3) PQ-Secure Signatures: To address quantum threats, NIST standardized three digital signature schemes under its PQC initiative [7]: ML-DSA (FIPS 204 [21]), FN-DSA [22], and SLH-DSA, (FIPS 205 [23]). In $\mathtt{QPADL}$, we adopt ML-DSA, based on Module-LWE, with $(sk, PK) \leftarrow \mathtt{ML\text{-}DSA.KeyGen}(1^\lambda)$, $\sigma \leftarrow \mathtt{ML\text{-}DSA.Sign}(sk, m)$, and $\{0,1\} \leftarrow \mathtt{ML\text{-}DSA.Verify}(PK, m, \sigma)$.

Ring signatures allow a user to anonymously sign a message on behalf of a group, while linkable ring signatures (LRSs) additionally enable detection of multiple signatures generated by the same signer under a defined context. In event-oriented LRS, an event identifier is embedded in each signature, allowing linkability across signatures with the same key and event-id without revealing the signer's identity [24].

Definition 3. *An $\mathtt{LRS}$ scheme consists of five algorithms:*

- $pp \leftarrow \mathtt{LRS.Gen}(1^\lambda)$*: Given the security parameter λ, it outputs the public parameters pp.*
- $(sk, PK) \leftarrow \mathtt{LRS.KeyGen}(pp)$*: On input pp, it returns a pair of private and public keys (sk, PK).*
- $\sigma \leftarrow \mathtt{LRS.Sign}(e_{ID}, sk_{l_r}, m, \mathsf{R})$*: Given event identifier e_{ID}, ring member's secret key sk_{l_r}, message m, and public key list R, it outputs a signature σ.*
- $\{0,1\} \leftarrow \mathtt{LRS.Verify}(e_{ID}, \sigma, m, \mathsf{R})$*: It takes e_{ID}, a signature σ on the message m, and a list of public keys R, and outputs 1 or 0 representing accept and reject.*
- $\{0,1\} \leftarrow \mathtt{LRS.Link}(e_{ID}, \sigma, \sigma', m, m', \mathsf{R}, \mathsf{R}')$*: Given e_{ID}, signatures (σ, σ') on messages (m, m'), and public key lists $(\mathsf{R}, \mathsf{R}')$, it returns 1 if linked, 0 otherwise.*

We adopt a PQ secure event-oriented LRS scheme [24] built from a hash function and Signature of Knowledge (SoK) primitives (e.g., STARK-based SoK (ethSTARK) [25]) with offline/online signing and verification. The signer proves in zero knowledge that (i) their public key is a leaf in a Merkle tree over the ring, and (ii) the tag is correctly computed from the secret key and event-id. Verification checks the event-id, Merkle root, and tag consistency. Linkability is enforced by matching tags across signatures sharing the same event-id.

4) PQ-variant of The Onion Routing ($\mathtt{PQ\text{-}Tor}$): With the advent of NIST-standardized PQC, the PQ variant of the Tor network ($\mathtt{PQ\text{-}Tor}$) retains the core architecture of conventional Tor while replacing vulnerable cryptographic components. Specifically, $AES128$ is upgraded to $AES256$ to mitigate Grover's algorithm [26]; RSA signatures used in consensus are replaced with $\mathtt{ML\text{-}DSA}$ [21]; RSA-based KEMs for circuit creation are substituted with $\mathtt{ML\text{-}KEM}$ [27]; and all key types (short-, medium-, and long-term) are updated to their PQ-secure equivalents. These substitutions preserve the functional design of Tor while achieving quantum resilience.

Communication in $\mathtt{PQ\text{-}Tor}$ proceeds as follows: *(i)* $\mathtt{PQ\text{-}Tor.Setup}$: The client connects to a Directory Authority (DA), retrieves the latest network state, and selects the relay path in reverse: exit node N_x, middle node N_m, entry node N_e. *(ii)* $\mathtt{PQ\text{-}Tor.Circuit.Creation}$: The client sends *CREATE* and *EXTEND* commands to establish the circuit, generating three AES keys and distributing them using $\mathtt{ML\text{-}KEM}$ encapsulation. *(iii)* $\mathtt{PQ\text{-}Tor.Send}(N_r; m)$ transmits message m to receiver N_r via layered symmetric encryption, while $\mathtt{PQ\text{-}Tor.Receive}(N_s; m)$ receives message m from sender

N_s. Each relay decrypts a layer and forwards the message, with the exit node ultimately delivering it to the destination.

III. Threat and Security Models

1) Threat Model: We consider a threat model with a quantum-capable adversary controlling the wireless medium, endangering confidentiality, authentication, and integrity during spectrum access. The adversary aims to compromise user location privacy and identity, launch DoS attacks on PSDs and SAS servers, and exploit the system through falsified location claims. PSDs and SAS servers are modeled as honest-but-curious, executing their roles correctly while attempting to infer sensitive user data such as location or identity. Accordingly, our threat model encompasses the following attack vectors:

- *Client Privacy and Anonymity:* The adversary (i.e., PSDs, SAS servers, or external observers) aims to compromise users' private information, specifically their precise geographic location, device-specific attributes, or real-world identity, by analyzing messages exchanged during the spectrum query and access phases. This includes passive observation, active probing, or correlating metadata to deanonymize users and infer sensitive location data.
- *Location Spoofing Attacks:* Users are required to provide their exact location but may behave maliciously by submitting false coordinates to gain unauthorized access to spectrum channels or SAS services outside their authenticated locations. They may also fall victim to spoofing or compromise. Potential location-based attacks include fake coordinates, proof replay, collusion with malicious entities, distance fraud, relay/mafia fraud, distance hijacking, and timestamp manipulation [11].
- *Denial-of-Service Attacks:* Malicious users or external adversaries may launch DoS attacks during spectrum queries or service requests to degrade the availability of SAS infrastructure. By flooding the PSD or servers with bogus, replayed, or computationally expensive requests, they aim to exhaust system resources, delay responses, or block legitimate spectrum access. These attacks, ranging from high-rate spamming to cryptographic overload, are especially damaging in distributed, latency-sensitive environments, where timely access to spectrum is critical.

2) Security Model: Given the system and threat models, QPADL aims to provide the following security objectives:

Definition 4 (t-Private PIR)**.** *A multi-server PIR with* DB *and security parameter λ is* t-private *if for every $0 < t < \ell$ and every adversary $\mathcal{A}$ corrupting any subset $S \subset [\ell]$ with $|S| \le t$, there exists a simulator $\mathcal{S}$ such that for all query indices $i_0, i_1 \in \mathtt{DB}$, $|\Pr[\mathcal{A}(\mathsf{View}_S^{PIR}(i_0)) = 1] - \Pr[\mathcal{A}(\mathsf{View}_S^{PIR}(i_1)) = 1]| \le \mathsf{negl}(\lambda)$, where $\mathsf{View}_S^{PIR}(i)$ denotes the joint view of the corrupted servers in S when the honest client queries index i.*

Definition 5 (ν-Byzantine-Robust PIR)**.** *A PIR is ν-Byzantine-robust if, for any set $B \subset [\ell]$ of at most ν Byzantine servers and any index $i \in \mathtt{DB}$, $\Pr[\mathtt{BlockReconst}_{PIR}(\rho_{[\ell]}(i, B)) = \mathtt{DB}[i]] = 1$, where $\rho_{[\ell]}(i, B)$ are the answers from all ℓ servers with those in B possibly adversarial.*

Definition 6 (k-out-of-ℓ PIR)**.** *A PIR scheme is k-out-of-ℓ correct if, for any index $\theta \in \mathtt{DB}$ and any subset $S \subseteq [\ell]$ with $|S| \ge k$, $\Pr[\mathtt{BlockReconst}_{PIR}(\rho_S(\theta)) = \mathtt{DB}[\theta]] = 1$, where $\rho_S(\theta)$ are the answers from servers in S.*

Definition 7 (PQ-Tor Anonymity)**.** *A PQ-Tor network provides anonymity if any quantum-capable adversary cannot distinguish, beyond negligible probability in λ, between two executions differing only in the honest sender (or receiver), assuming all layers use PQ-secure KEMs and symmetric ciphers.*

Definition 8 (Correctness and Soundness of Location Verification)**.** *A location verification scheme is* correct *if, for any honest user at (l_x, l_y), the proof of location (PoL) verifies with probability* $\Pr[\mathtt{Verify}(\mathtt{PoL}(l_x, l_y)) = 1] \ge 1 - \mathsf{negl}(\lambda)$. *It is* sound *if, for any PPT adversary $\mathcal{A}$ and any $l' \neq l$,* $\Pr[\mathtt{Verify}(\mathtt{PoL}') = 1 \wedge \mathsf{Loc}(\mathtt{PoL}') = l'] \le \mathsf{negl}(\lambda)$ *unless $\mathcal{A}$ is physically present at l'. Proofs are bound to spatio-temporal context, non-transferable, and non-replayable via cryptographic commitments and signatures, resisting relay, distance, mafia, and hijacking attacks.*

Definition 9 (Counter-DoS)**.** *A system is DoS-resilient if for every PPT adversary $\mathcal{A}$ issuing at most $q(\lambda)$ queries, the probability that $\mathcal{A}$ causes unavailability, defined as delaying any honest request beyond a fixed bound Δ, without expending computational cost $\Omega(q(\lambda) \cdot \tau)$ is at most* $\mathsf{negl}(\lambda)$, *where τ is the per-query puzzle cost. Formally,* $\Pr\left[\mathsf{Delay}(\mathcal{A}) > \Delta \wedge \mathsf{Cost}(\mathcal{A}) < q(\lambda) \cdot \tau\right] \le \mathsf{negl}(\lambda)$, *with τ enforced via rate-limiting, client puzzles, and authenticated queries.*

3) Scope of Our Solution: QPADL framework preserves the location privacy and anonymity of SUs during spectrum access. It protects users while querying spectrum availability, retrieving cryptographic puzzles, and requesting services, offering security against identity and location disclosure, location spoofing, and DoS, while ensuring PQ security. This work focuses solely on the spectrum query and access phases. It does not address privacy risks during user registration or spectrum utilization. PUs are outside the scope of this work. Additionally, the framework does not cover location leakage during spectrum usage, user mobility, or handover scenarios [28]. It also excludes timing attacks, side-channel leaks, and signal localization techniques (e.g., triangulation) that adversaries may use to determine SU's locations [29].

IV. The Proposed Framework: QPADL

We outline the initial setup of the QPADL framework, followed by its main operations, detailed algorithmic description, and various instantiations and optimization strategies.

A. QPADL Framework Architecture and Initial Setup

The initial setup of QPADL establishes the geolocation database, configures APs, and initializes the PQ-Tor network.

1) DB Structure and Setup: The DB structure in QPADL adheres to FCC spectrum-sharing regulations, with each PSD storing synchronized entries indexed by a subroutine DB.Index(.) that maps tuples $((l_x, l_y), ch, \mathtt{TV})$, which consist of grid-based coordinates, frequency channel, and time validity window, to specific DB blocks. Each entry also includes parameters such as maximum transmission power (EIRP) and spectrum data [8]. The indexing can be instantiated using FCC-compliant location encoding methods such as geohash (base32), H3 (hexagonal grids), or S2 geometry (Google SAS) to enable efficient spatial queries [30]. The DB is modeled as an $r_{\mathtt{DB}} \times s_{\mathtt{DB}}$ matrix, where each row is a b-bit block partitioned into $s_{\mathtt{DB}}$ words over $GF(2)$. Indexing is driven by location, device attributes (e.g., power, height, type), and access preferences (e.g., bandwidth, duration), ensuring consistency and integrity across all synchronized DBs. The `DB.Record` stores the associated puzzle and signature at the resolved location computed by the `DB.Index` subroutine for a given index.

2) Access Points Setup: APs in a region collectively form a ring R containing the PKs of all participants, where each AP holds a key pair $(sk_{\mathtt{AP}}, PK_{\mathtt{AP}})$ generated by the FCC using `LRS.KeyGen`(pp) algorithm (Definition 3). They periodically broadcast time-sensitive beacons $(\beta_{\mathtt{TS}})$ for device discovery within a time window (`TS`). To estimate a user's proximity, each AP applies signal strength and RTT analysis based on environmental parameters [31], [32], computing the distance via $\Delta \leftarrow \mathtt{ProxVerif}(RSS, RTT, env_{params})$ algorithm.

3) `PQ-Tor` Setup and Configuration: We assume that PQ-Tor has been initialized and is ready for use via `PQ-Tor.Setup`. The client proceeds by establishing an anonymous communication circuit using `PQ-Tor.Circuit.Creation`, followed by data transmission through `PQ-Tor.Send` and `PQ-Tor.Receive`, as detailed in Section II-B4.

B. `QPADL` Framework Main Operations

The overall flow of the `QPADL` framework is presented in Algorithms 1-2, detailing its core operations as outlined below:

Algorithm 1 `QPADL` Framework

```
❶ $\mathtt{DB} \leftarrow \mathtt{PSD.Puzzle.Bind(DB)}$:
 1: for all $(l_x, l_y)$ do
 2:     for all $ch$ do
 3:         for all $\mathtt{TV}$ do
 4:             Given $\theta \leftarrow \mathtt{DB.Index}((l_x, l_y), ch, \mathtt{TV})$
 5:             $\pi_\theta \leftarrow \mathtt{Puzzle.Gen}(1^\lambda, \kappa)$
 6:             $\sigma_{\pi_\theta} \leftarrow \mathtt{ML\text{-}DSA.Sign}(sk_{\mathtt{PSD}}, \pi_\theta)$
 7:             $\mathtt{DB.Record}(\pi_\theta, \sigma_{\pi_\theta})$
❷ $\{q_i\}_{i=1}^n \leftarrow \mathtt{Client.Spectrum.Query}((l_x, l_y), ch, \mathtt{TV}, \mathtt{TS}, \beta_{\mathtt{TS}})$:
 8: $(C_{\mathtt{TS}}, \sigma_{\mathtt{AP}}, e_{\mathtt{ID}}) \leftarrow \mathtt{Client.PoL}((l_x, l_y), \mathtt{TS}, \beta_{\mathtt{TS}})$
 9: $\theta \leftarrow \mathtt{DB.Index}((l_x, l_y), ch, \mathtt{TV})$
10: $(q_1, q_2, ..., q_n) \leftarrow \mathtt{PIR.Query}(\theta)$
11: for $i = 1, \ldots, n$ do
12:     $\mathtt{PQ\text{-}Tor.Send}(\mathtt{PSD}_i; q_i, C_{\mathtt{TS}}, \sigma_{\mathtt{AP}}, e_{\mathtt{ID}})$
13: $\mathtt{PSD.PQ\text{-}Tor.Receive}(\mathtt{SU}; q_i, C_{\mathtt{TS}}, \sigma_{\mathtt{AP}}, e_{\mathtt{ID}})$
❸ $\rho_i \leftarrow \mathtt{PSD.Spectrum.Response}(q_i, C_{\mathtt{TS}}, \sigma_{\mathtt{AP}}, e_{\mathtt{ID}}, R)$:
14: if $1 \leftarrow \mathtt{PoL.Verify}(e_{\mathtt{ID}}, \sigma_{\mathtt{AP}}, C_{\mathtt{TS}}, R)$ then
15:     if $1 \leftarrow \mathtt{LRS.Link}(e_{\mathtt{ID}}, \sigma_{\mathtt{AP}}, C_{\mathtt{TS}}, R, \vec{\sigma}_{\mathtt{AP}})$ then, return $\perp$
16:     else
17:         Record $\sigma_{\mathtt{AP}}$
18:         $\rho_i \leftarrow \mathtt{PIR.Query.Response}(q_i, \mathtt{DB})$
19: $\mathtt{PSD.PQ\text{-}Tor.Send}(\mathtt{SU}; \rho_i)$
20: for $i = 1, \ldots, k$, where $k \in [t, n]$ do
21:     $\mathtt{Client.PQ\text{-}Tor.Receive}(\mathtt{PSD}_i; \rho_i)$
22: $\mathtt{DB}_\theta \leftarrow \mathtt{Client.PIR.BlockReconst}(\theta, (\rho_1, \rho_2, ..., \rho_k))$
❹ $Token \leftarrow \mathtt{Client.Create.Token}(\Pi, \sigma_\Pi)$:
23: if $1 \leftarrow \mathtt{ML\text{-}DSA.Verify}(PK_{\mathtt{PSD}}, \Pi, \sigma_\Pi)$ then
24:     $\Psi \leftarrow \mathtt{PoW}(\Pi, \kappa)$
25:     return $Token \leftarrow (\Pi, \sigma_\Pi, \Psi)$
❺ $\{0,1\} \leftarrow \mathtt{Client.Service.Request}(C_{\mathtt{TS}}, \sigma_{\mathtt{AP}}, e_{\mathtt{ID}})$:
26: if $1 \leftarrow \mathtt{ML\text{-}DSA.Verify}(PK_{\mathtt{PSD}}, \sigma_\Pi)$, then
27:     if $1 \leftarrow \mathtt{PoW.Verify}(\Pi, \Psi)$ then, Record the $Token$
28:         if $1 \leftarrow \mathtt{PoL.Verify}(C_{\mathtt{TS}}, e_{\mathtt{ID}}, \sigma_{\mathtt{AP}}, R)$, then
29:             return 1, and grant access.
```

*1) **Puzzle Management and Private Spectrum Services:*** `PSD`s initialize and populate the DBs with spectrum information (e.g., location, available channels) and signed puzzles. For each grid segment, defined by coordinates, time frames, and device attributes (e.g., power level, height, category), puzzles are generated at appropriate difficulty levels and signed using `ML-DSA.Sign`. Entries are indexed as $\mathtt{DB.Index} \leftarrow ((l_x, l_y), ch, \mathtt{TV})$ and stored in `DB`, with puzzles and signatures refreshed periodically based on validity intervals (e.g., hourly). The number of puzzles generated depends on device characteristics, server count, and their maximum capacity [8].

*2) **Proof of Location (`PoL`) Acquisition and Validation:*** In this phase, the client obtains a valid `PoL` for a specific time and geographic area (Algorithm 2). After receiving periodic beacons $(\beta_{\mathtt{TS}})$ from nearby APs, the client selects the AP with the strongest signal and requests a `PoL` for the current window (`TS`), sending a commitment $(C_{\mathtt{TS}})$ to the latest beacon and obfuscated coordinates with a random nonce r to protect location privacy (Steps 1-4). The AP validates the request against its latest beacon and checks proximity using signal strength and RTT (Steps 5-7). If successful, it derives an event-id from $sk_{\mathtt{AP}}$, $\beta_{\mathtt{TS}}$, and `TS`, generates a ring signature on the user's commitment, and returns both to the client (Steps 8-11). The client verifies the signature and accepts it as a valid proof of location (Step 12).

*3) **Querying Spectrum Availability and Retrieving Quantum Safe Puzzles:*** To comply with FCC regulations and participate in the counter-DoS mechanism, clients retrieve puzzles and spectrum information from `PSD`s after obtaining a `PoL` for the current time window (Step 8). Using their location, channel, and timestamp, they compute the target index θ and generate queries for each `PSD` (Steps 9-10), which are sent with the `PoL` via `PQ-Tor` (Steps 11-12). Each `PSD` verifies the `PoL` and checks linkability with prior `PoL`s in the same window, rejecting linked queries to prevent DoS. Otherwise, it records the `PoL`, computes the response, and returns it through `PQ-Tor`. The client then reconstructs the target item from the responses (Steps 20-22).

Algorithm 2 $(C_{\mathtt{TS}}, \sigma_{\mathtt{AP}}, e_{\mathtt{ID}}) \leftarrow \mathtt{PoL}((l_x, l_y), \mathtt{TS}, \beta_{\mathtt{TS}})$

```
❶ $C_{\mathtt{TS}} \leftarrow \mathtt{Client.PoL.Request}(\beta_{\mathtt{TS}})$:
 1: Given the latest beacon $\beta_{\mathtt{TS}}$
 2: $r \xleftarrow{\$} \mathbb{Z}_q$
 3: $C_{\mathtt{TS}} \leftarrow H((l_x, l_y)||\beta_{\mathtt{TS}}||\mathtt{TS}||r)$
 4: Send $(\beta_{\mathtt{TS}}, C_{\mathtt{TS}})$ to $\mathtt{AP}$.
❷ $(\sigma_{\mathtt{AP}}, e_{\mathtt{ID}}) \leftarrow \mathtt{AP.PoL.Response}(\beta_{\mathtt{TS}}, C_{\mathtt{TS}})$:
 5: if $\beta_{\mathtt{TS}}$ is not the latest beacon: then, return $\perp$
 6: else
 7:     $\Delta_c \leftarrow \mathtt{ProxVerif}(RSS, RTT, env_{params})$
 8:     if $\Delta_c \leq \Delta_{th}$ then
 9:         $e_{\mathtt{ID}} \leftarrow H'(sk_{\mathtt{AP}}, \beta_{\mathtt{TS}}, \mathtt{TS})$
10:         $\sigma_{\mathtt{AP}} \leftarrow \mathtt{LRS.Sign}(e_{\mathtt{ID}}, C_{\mathtt{TS}}, sk_{\mathtt{AP}})$
11: Send $(\sigma_{\mathtt{AP}}, e_{\mathtt{ID}})$ to the Client.
❸ $\{0,1\} \leftarrow \mathtt{Client.PoL.Verify}(\sigma_{\mathtt{AP}}, e_{\mathtt{ID}}, C_{\mathtt{TS}}, R)$:
12: if $1 \leftarrow \mathtt{LRS.Verify}(e_{\mathtt{ID}}, \sigma_{\mathtt{AP}}, C_{\mathtt{TS}}, R)$ then, return 1
```

*4) **Token Creation and/or SAS Service Request:*** After obtaining frequency availability or SAS service details and verifying the puzzle's signature (Step 23), the client solves the puzzle via brute force and generates a token for server communication (Steps 24-25). The client then submits a request with the token for a specified time interval. The server verifies the puzzle's authenticity using the `PSD`'s signature (Step 26) and then validates the token (Step 27). If valid, the server: *(i)* records the token to prevent DoS (Step 28), *(ii)* verifies the `PoL` for location commitment (Step 29), *(iii)* requests the client to reveal the commitment and checks $C = H((l_x, l_y)||\beta||\texttt{TS})$, and *(iv)* grants service access.

C. `QPADL` Framework Instantiations

To meet the design and security requirements of the `QPADL` framework, we propose three PIR instantiations for the spectrum query phase and two PoW instantiations for the service request phase. We then present a full instantiation of `QPADL`, detailing its step-by-step flow as shown in Fig. 1.

*1) **PIR Instantiations**:* While any t-private PQ-secure PIR can be employed in `QPADL`, we introduce three tailored instantiations, each offering distinct security features and trade-offs.

(i) `QPADL-ENS`: This is the most efficient instantiation for the query phase, based on the IT-secure PIR scheme from [18], which assumes n non-colluding, responsive servers maintaining synchronized database copies, hence named ENS (Efficient Non-colluding Servers). The client performs only XOR operations over random r-bit vectors, while each `PSD` performs a single multiplication of the query over the entire database matrix. The PIR algorithm is described in [18], with a GPU-parallelized version detailed in Section IV-D2.

(ii) `QPADL-FTR`: This Fault Tolerant Robust (FTR) instantiation of `QPADL` adopts the IT-secure PIR scheme from [19], enabling ν-Byzantine fault tolerance by allowing block reconstruction even if up to ν servers return incorrect responses. Servers compute a query vector multiplication over the database matrix. Using Shamir's secret sharing, the client can reconstruct the target block via Lagrange interpolation when responses are received from any k out of $[t, n]$ `PSD`s. In cases of synchronization errors, transmission faults, or Byzantine behavior ($\nu < k$), the client employs Guruswami-Sudan list decoding for error correction, ensuring robustness. Further details of this PIR scheme are provided in [19], with its parallelized implementation described in Section IV-D3.

(iii) `QPADL-OOP`: This instantiation, named for its use of Online-Offline Preprocessing (OOP), maximizes efficiency and enhances DB structure while providing computational security. It employs CIP-PIR [33], a PQ-secure PIR protocol optimized for large-scale networks with multi-GB DBs where servers hold identical but not strictly replicated data. Improving upon Chor et al. [18], it uses seed-based queries to reduce communication and restricts each server's data access to a subset, minimizing online computation. Unlike Chor's model, seed selection occurs server-side during preprocessing. CIP-PIR includes two preprocessing steps: a one-time DB preprocessing during setup and a client-independent step that precomputes response components unrelated to any specific query, enabling reuse for a single future query. This setup assumes `PSD`s maintain the same data, though chunk order may vary. Due to its performance-security trade-offs, this instantiation and its algorithmic flow are included in our full framework instantiation depicted in Fig. 1.

*2) **PoW Instantiations**:* `QPADL` supports various PQ-secure PoW mechanisms, with instantiations shown below:

(i) `QPADL-HCT`: Given the burden of token creation on the users, this instantiation adopts the Hashcash Tree (`HCT`) construction to provide an efficient and PQ-secure PoW mechanism, considering users' diverse computational resources (e.g., CPU, energy, battery, storage). As an enhanced version of the original Hashcash puzzle used in Bitcoin [34], `HCT` is designed to resist quantum and parallel brute-force attacks [35], [36], offering adjustable difficulty tailored to varying user capabilities. Built on hash functions, it inherently ensures efficiency in both classical and PQ settings, leveraging sequential hardness via a binary tree structure that prevents full parallel shortcuts. The detailed `HCT` algorithm is presented below:

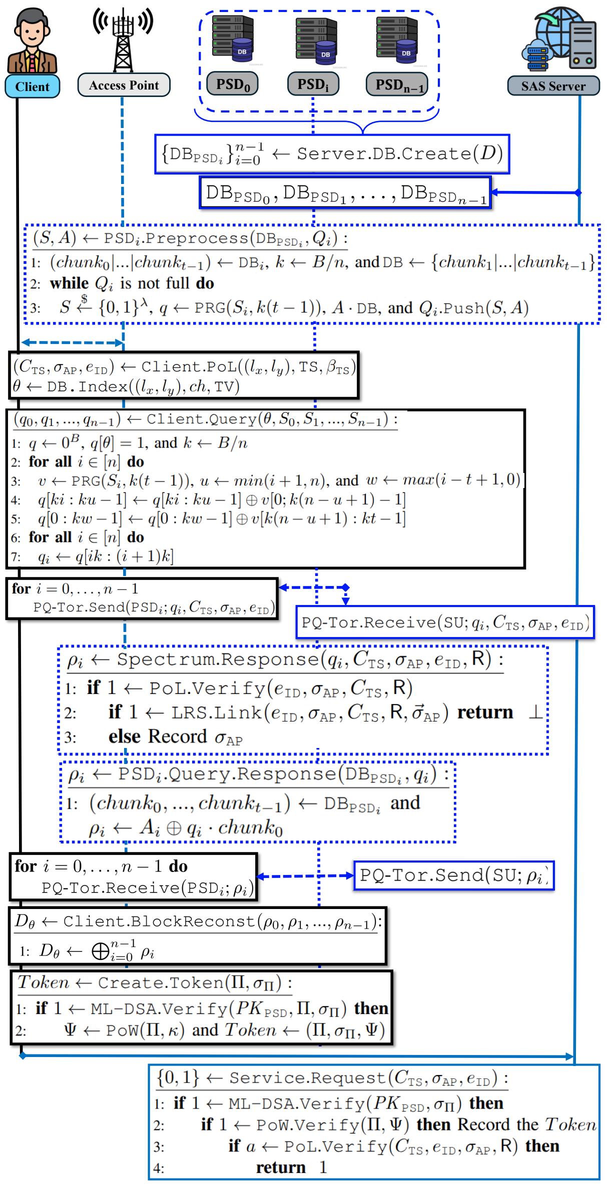
- $\Pi \leftarrow$ `HCT.Puzzle.Gen`$(1^\lambda, \kappa)$: Randomly selects $n_s \overset{\$}{\leftarrow} \{0,1\}^\lambda$ and sets the number of leaves n_l based on the difficulty level κ. The puzzle is $\Pi = (h, n_s, \kappa, n_l)$.
- $\Psi \leftarrow$ `HCT.PoW`(Π): Constructs a perfect binary tree of Hashcash puzzles via brute force. For each leaf node ($i > n_l$), the client finds n_x such that $h_\kappa(n_s||i||0||0||n_x)$ has κ leading zeros. For internal nodes ($i < n$), it solves $h_\kappa(n_s||i||h_{2i}||h_{2i+1}||n_x)$. The root's nonce n_1 is committed as the PoW token, with all nonces being $(n_{2^{n_l}-1}, \ldots, n_1)$.
- $\{0,1\} \leftarrow$ `HCT.PoW.Verify`(Π, Ψ): Upon receiving the root's nonce, the server randomly selects a leaf index $i \in [1, n_l]$ and requests the corresponding path from leaf to root. Verification requires only $\log n$ hash computations.

(ii) `QPADL-LBP`: An alternative PoW instantiation for `QPADL` leverages the lattice-based construction from [37], built on the hardness of the Shortest Vector Problem (SVP) [38]. This approach offers quantum-edge security while it introduces higher computational and communication overhead. It comprises three core subroutines, described below:

- $\Pi \leftarrow$ `LB.Puzzle.Gen`$(1^\lambda, 1^{n_\Lambda}, \kappa)$: Given security parameter λ, lattice dimension n_Λ, and difficulty level κ, the algorithm samples $x_2, \ldots, x_n \leftarrow \mathcal{U}(0 \cup [p-1])$, sets $\alpha = 1.05 \cdot \Gamma(n/2+1)^{1/n}/\sqrt{\pi}$, and constructs the lattice basis which is an $n_\Lambda \times n_\Lambda$ matrix B with the first row comprised of $[p\ x_2\ \ldots\ x_n]$, ones on the subdiagonal, and zeros elsewhere. It outputs the puzzle $\Pi = (\alpha, n_\Lambda, B, p)$.
- $\Psi \leftarrow$ `LB.PoW`(Π, κ): Solves Π by finding $v \in \Lambda(B)$ such that $||v|| \leq \alpha \cdot p^{1/n_\Lambda}$, and returns $\Psi = (v, \nu)$ where $v = B \cdot \nu$.
- $\{0,1\} \leftarrow$ `LB.PoW.Verify`(Π, Ψ): Verifies the solution by checking $||v|| \leq \alpha \cdot p^{1/n_\Lambda}$, $v = B \cdot \nu$, and $\nu \in \mathbb{Z}^{n_\Lambda}$, returning 1 if all conditions hold.

*3) **Full Instantiations of `QPADL`**:* Fig.1 presents the full operational flow of PIR and PoW instantiations in `QPADL`, using CIP-PIR for spectrum queries and `HCT` puzzles for PoW. The `DB` contains frequency data $((l_x, l_y), ch, \texttt{TV})$, with each entry bound to a signed puzzle $(\pi_\theta, \sigma_{\pi_\theta})$ via `Puzzle.Bind()` (Algorithm1, Steps 1-7). While all `PSD`s store the same content, their chunk orders may differ. Each block $D_\theta = ((l_x, l_y), ch, \texttt{TV}, \Pi_\theta, \sigma_{\Pi_\theta})$ is divided into B chunks, with $k = B/n$. For all $i \in [n]$, chunks are defined as $chunk_i = block_{ki}|...|block_{ki+k-1}$, and each `PSD` arranges its DB as $\texttt{DB}_{\texttt{PSD}_i} = chunk_i|chunk_{i+1 \mod n}|...|chunk_{i+t-1 \mod n}$.

The CIP-PIR scheme requires each $\texttt{PSD}_i$ to perform a one-time, client-independent preprocessing on its local DB, generating seed-value pairs (S, A) and storing them in a local queue Q_i. Specifically, for each DB chunk, a randomly chosen


Client
Access Point
$\mathbf{PSD_0}$
$\mathbf{PSD_i}$
$\mathbf{PSD_{n-1}}$
SAS Server
DB
$\{\texttt{DB}_{\texttt{PSD}_i}\}_{i=0}^{n-1} \leftarrow \texttt{Server.DB.Create}(D)$
$\texttt{DB}_{\texttt{PSD}_0}, \texttt{DB}_{\texttt{PSD}_1}, \ldots, \texttt{DB}_{\texttt{PSD}_{n-1}}$

```
$(S, A) \leftarrow \texttt{PSD}_i\texttt{.Preprocess}(\texttt{DB}_{\texttt{PSD}_i}, Q_i)$ :
1: $(chunk_0|...|chunk_{t-1}) \leftarrow \texttt{DB}_i$, $k \leftarrow B/n$, and $\texttt{DB} \leftarrow \{chunk_1|...|chunk_{t-1}\}$
2: while $Q_i$ is not full do
3:     $S \xleftarrow{\$} \{0,1\}^\lambda$, $q \leftarrow \texttt{PRG}(S_i, k(t-1))$, $A \cdot \texttt{DB}$, and $Q_i\texttt{.Push}(S, A)$
```

```
$(C_{\texttt{TS}}, \sigma_{\texttt{AP}}, e_{\texttt{ID}}) \leftarrow \texttt{Client.PoL}((l_x, l_y), \texttt{TS}, \beta_{\texttt{TS}})$
$\theta \leftarrow \texttt{DB.Index}((l_x, l_y), ch, \texttt{TV})$
```

```
$(q_0, q_1, ..., q_{n-1}) \leftarrow \texttt{Client.Query}(\theta, S_0, S_1, ..., S_{n-1})$ :
1: $q \leftarrow 0^B$, $q[\theta] = 1$, and $k \leftarrow B/n$
2: for all $i \in [n]$ do
3:     $v \leftarrow \texttt{PRG}(S_i, k(t-1))$, $u \leftarrow min(i+1, n)$, and $w \leftarrow max(i-t+1, 0)$
4:     $q[ki : ku-1] \leftarrow q[ki : ku-1] \oplus v[0; k(n-u+1)-1]$
5:     $q[0 : kw-1] \leftarrow q[0 : kw-1] \oplus v[k(n-u+1) : kt-1]$
6: for all $i \in [n]$ do
7:     $q_i \leftarrow q[ik : (i+1)k]$
```

```
for $i = 0, \ldots, n-1$
    $\texttt{PQ-Tor.Send}(\texttt{PSD}_i; q_i, C_{\texttt{TS}}, \sigma_{\texttt{AP}}, e_{\texttt{ID}})$
```

```
$\texttt{PQ-Tor.Receive}(\texttt{SU}; q_i, C_{\texttt{TS}}, \sigma_{\texttt{AP}}, e_{\texttt{ID}})$
```

```
$\rho_i \leftarrow \texttt{Spectrum.Response}(q_i, C_{\texttt{TS}}, \sigma_{\texttt{AP}}, e_{\texttt{ID}}, \mathsf{R})$ :
1: if $1 \leftarrow \texttt{PoL.Verify}(e_{\texttt{ID}}, \sigma_{\texttt{AP}}, C_{\texttt{TS}}, \mathsf{R})$
2:     if $1 \leftarrow \texttt{LRS.Link}(e_{\texttt{ID}}, \sigma_{\texttt{AP}}, C_{\texttt{TS}}, \mathsf{R}, \vec{\sigma}_{\texttt{AP}})$ return $\perp$
3:     else Record $\sigma_{\texttt{AP}}$
```

```
$\rho_i \leftarrow \texttt{PSD}_i\texttt{.Query.Response}(\texttt{DB}_{\texttt{PSD}_i}, q_i)$ :
1: $(chunk_0, ..., chunk_{t-1}) \leftarrow \texttt{DB}_{\texttt{PSD}_i}$ and
   $\rho_i \leftarrow A_i \oplus q_i \cdot chunk_0$
```

```
for $i = 0, \ldots, n-1$ do
    $\texttt{PQ-Tor.Receive}(\texttt{PSD}_i; \rho_i)$
```

```
$\texttt{PQ-Tor.Send}(\texttt{SU}; \rho_i$
```

```
$D_\theta \leftarrow \texttt{Client.BlockReconst}(\rho_0, \rho_1, ..., \rho_{n-1})$:
1: $D_\theta \leftarrow \bigoplus_{i=0}^{n-1} \rho_i$
```

```
$Token \leftarrow \texttt{Create.Token}(\Pi, \sigma_\Pi)$ :
1: if $1 \leftarrow \texttt{ML-DSA.Verify}(PK_{\texttt{PSD}}, \Pi, \sigma_\Pi)$ then
2:     $\Psi \leftarrow \texttt{PoW}(\Pi, \kappa)$ and $Token \leftarrow (\Pi, \sigma_\Pi, \Psi)$
```

```
$\{0,1\} \leftarrow \texttt{Service.Request}(C_{\texttt{TS}}, \sigma_{\texttt{AP}}, e_{\texttt{ID}})$ :
1: if $1 \leftarrow \texttt{ML-DSA.Verify}(PK_{\texttt{PSD}}, \sigma_\Pi)$ then
2:     if $1 \leftarrow \texttt{PoW.Verify}(\Pi, \Psi)$ then Record the $Token$
3:         if $a \leftarrow \texttt{PoL.Verify}(C_{\texttt{TS}}, e_{\texttt{ID}}, \sigma_{\texttt{AP}}, \mathsf{R})$ then
4:             return 1
```

Fig. 1: `QPADL` Full Instantiation

seed S is expanded into a $k(t-1)$-bit query q using a pseudo-random generator (`PRG`) (Steps 1-2). The corresponding value A is computed by XORing all non-flip chunks whose positions in q are set to 1, and the resulting (S, A) pair is added to Q_i (Step 3). Since each `PSD` holds $(t-1)$ non-flip chunks out of n, the preprocessing phase covers $(t-1)/n$ of the DB, leaving only $1/n$ for computation during the online phase.

To access spectrum, clients must first obtain a valid proof of location in phase two. After receiving the latest beacon $\beta_{\texttt{TS}}$ from a nearby AP, the client sends a location commitment as a `PoL` request. If the AP verifies proximity, it derives an event identifier from $\beta_{\texttt{TS}}$ and `TS`, signs the commitment, and returns $(\sigma_{\texttt{AP}}, e_{\texttt{ID}})$ (Algorithm 2, Steps 1-12). The client then computes the index via `DB.Index` and proceeds with the PIR query. It constructs a zero-initialized b-bit vector with the θ-th bit set, expands each `PSD`'s seed S_i into a $k(t-1)$-bit vector v using a `PRG`, and XORs v into the corresponding chunks of the query. The final query is partitioned into n sub-queries q_i, each mapped to the designated chunk of server $\texttt{PSD}_i$ (Steps 1-7). Finally, the client privately transmits the location proof and queries $(q_i, C_{\texttt{TS}}, \sigma_{\texttt{AP}}, e_{\texttt{ID}})$ to each $\texttt{PSD}_i$ through `PQ-Tor`.

Upon receiving a query, each `PSD` validates `PoL` with `PoL.Verify` (Algorithm 1, Steps 14-15) and applies `LRS.Link` to detect reuse within the same time window. Replayed proofs are rejected; otherwise, the proof is logged and the PIR response is generated. Specifically, $\texttt{PSD}_i$ retrieves its assigned flip chunk (e.g., $chunk_0$ in $\texttt{DB}_i$), XORs the blocks indicated by set bits in q_i, and combines the result with the precomputed value A_i from the offline phase. Since only one chunk ($1/n$ of the DB) is accessed online, with the remaining $(t-1)/n$ processed beforehand, computation is minimized. The response is then sent back via the established `PQ-Tor`.

Upon receiving responses, the client reconstructs the target block via `BlockReconst` and verifies the `ML-DSA` signature on the retrieved puzzle. It then solves the puzzle with `HCT.PoW`, derives the root solution ψ, and forms the token $Token \leftarrow (\Pi, \sigma_\Pi, \Psi)$. To access a SAS service, the client submits its location commitment, `PoL`, and the token. The server verifies the `PSD`'s puzzle signature and the PoW solution; invalid or reused solutions are rejected as DoS attempts. Valid tokens are logged to prevent replay, and access is granted after confirming the proof of location. In certain services or when client behavior is suspicious, the server may request disclosure of the committed location, assumed to occur over a secure authenticated channel (e.g., PQ-TLS), which is beyond this work's scope.

D. Instantiation via Parallelization

Our design rationale is to improve the online server delay, especially when several queries are invoked by multiple users. Numerous GPU-accelerated works [33], [39], [40] have been proposed to improve the efficiency of PIR schemes. Gunther et al. [33] propose CIP-PIR, where GPU acceleration is leveraged to improve offline server computation, but without addressing online server runtime, which is directly related to response delay. [40] attempts to improve the efficiency of homomorphic encryption-based single-server PIR, and therefore do not align with our design rationale to provide resiliency against rogue attacks and single root of trust. Herein, we introduce the NVIDIA GPU architecture. We then describe our parallelized PIRs, which can be of independent interest beyond our use case to anonymous spectrum access.

1) NVIDIA GPU Architecture and CUDA: A Graphical Processing Unit (GPU) is designed to accelerate computationally intensive tasks by leveraging Single Instruction Multiple Threads (SIMT) execution. An NVIDIA GPU comprises multiple Stream Multiprocessors (SMs), each of which manages several CUDA cores. The latter executes general-purpose computations, where each operates at a base clock frequency (e.g., 1320 MHz). The circuit provides hierarchical memory types: *(i) global memory (≈GBs)*: can be accessed by all threads in SMs. It is an off-chip memory and the data transfer medium between system memory and GPU with high access throughput. *(ii) shared memory (≈KBs)*: is shared among cores in a single SM and provides a faster memory access compared to (i). *(iii) registers*: reside in each core, with the fastest memory access to hold the frequently accessed and local data. CUDA is an interface developed by NVIDIA [41], which allows programmers to define and execute operations on GPUs. A CUDA program launches a *kernel* over a multidimensional *grid* of blocks. Each *block* contains multiple *warps* (that is, the warp comprises 32 threads) running under the SIMT paradigm.

2) Parallel Chor-PIR: The online $\texttt{PSD}$ computational overhead consists of matrix multiplication over $GF(2)$ (i.e., $\rho \leftarrow q \cdot \texttt{DB}$), where $\rho \in \{0,1\}^b$, $\texttt{DB} \in \{0,1\}^{r\times b}$, and $q \in \{0,1\}^r$. This operation consists of conditional aggregation (via bitwise XOR) of matrix rows $\texttt{DB}$ based on the query's bit values $\{q_{r'}\}_{r'\in\{1,\ldots r\}}$. The total computational work per query is quantified as $\mathbb{W} = nnz(q) \cdot b$, where $nnz(q)$ denotes the Hamming weight of the query. The total memory traffic is equal to $\mathbb{Q} \leftarrow r + nnz(\rho) \cdot b + b$, accounting for the load of ρ and active rows of D, as well as the store of the response R. Consequently, the server-side computation is a memory-bound task due to its low operational intensity, formalized as $\mathbb{I} = \frac{\mathbb{W}}{\mathbb{Q}}$. Zhang et al. [42] demonstrate that Boolean linear algebra is not suited for GPU tensor cores. Instead, optimization efforts must be pivoted towards CUDA cores using memory hierarchy exploitation, data locality, and coalesced memory access.

Algorithm 3 Multi-request Parallel Chor-PIR

```
    $\underline{\boldsymbol{\rho} \leftarrow \texttt{PSD.Query.Response}(\boldsymbol{q})}$: Upon receiving multiple re-
    quests $\boldsymbol{q} = \{q_i = \{q_{i,1}, q_{i,2}, \ldots, q_{i,r}\}\}_{i=1}^{\bar{\boldsymbol{q}}} \in_R GF(2)^r$
 1: each $\texttt{PSD}_i$ for $i = 1, 2, ..., \ell$ do
 2:     copy $\boldsymbol{q}$ from main memory to global memory
 3:     $|grid| = (|\bar{\boldsymbol{q}}, \lfloor\frac{b}{32}\rfloor)$ , $|block| = (32, 8)$
 4:     CUDA Kernel:
 5:         for each block $(b_x, b_y) \in \{(1,1), \ldots, |grid|\}$ do
 6:             for each thread $(t_x, t_y) \in \{(1,1), \ldots, |block|\}$ do
 7:                 $c \leftarrow 32 \cdot b_y + t_x$ , $a \leftarrow 0$
 8:                 for $r' = t_y, \ldots, r$ [step $= 8$] do
 9:                     if $\boldsymbol{q}_{b_x,r'} = 1$ then $a \leftarrow a + \texttt{DB}_{r',c}$
10:                 store $a$ into shared memory: $a_{t_x,t_y} \leftarrow a$
11:                 synchronize threads in the block $(b_x, b_y)$
12:                 if $t_y = 0$ then
13:                     $\rho_{b_x,c} \leftarrow \bigoplus_{r'=1}^{8} a_{t_x,r'}$
14:                     write $\rho_{b_x,c}$ from register to global memory
15:     return $\boldsymbol{\rho} = \{\rho_i = \{\rho_{i,1}, \ldots, \rho_{i,b}\}\}_{i=1}^{\bar{\boldsymbol{q}}}$
```

Our proposed parallel Chor-PIR is detailed in Algorithm 3. $\texttt{Query.Response}$ processes mutliple queries $\boldsymbol{q} = \{q_i\}_{i=1}^{\bar{\boldsymbol{q}}}$ in batches. First, it copies the queries $\boldsymbol{q}$ to global memory before invoking the CUDA kernel (Step 2). The $\texttt{DB}$ is already residing in the global memory. The kernel is configured with $(\bar{\boldsymbol{q}}, \lfloor\frac{b}{32}\rfloor)$ blocks, each is configured as a $(32, 8)$ thread layout to enable warp-level parallelism and strided row-wise accumulation. Each thread (t_x, t_y) computes partial $GF(2)$ XOR accumulations on rows using an 8-stride loop. In particular, Step 9 introduces warp divergence due to the conditional row fetch based on $q_{b_x,r'}$. However, empirical profiling shows that the high memory throughput of this memory-bound kernel compensates for this divergence. One can use a bitmasking approach to avoid this divergence, but it incurs inferior performance, especially when q_i is a sparse vector (i.e., $nnz(q_i) \ll r$). The intermediate accumulations are written to shared memory for intra-level reduction. Threads with $ty = 0$ finalize the response by aggregating the 8 per-thread vertical sub-sums $\{a_{t_x,r'}\}_{r'}$, and committing the result to the global memory.

3) Parallel Goldberg-PIR: Algorithm 4 outlines the GPU-accelerated implementation of the Goldberg PIR. It operates in a batched setting, each containing a vector of queries $\boldsymbol{q}$, each processed over a database $\texttt{DB}$. $\boldsymbol{q}$ and $\texttt{DB}$ are of size $\bar{\boldsymbol{q}} \cdot r$ and $r \cdot b$, respectively. For each incoming PIR query batch $\boldsymbol{q}$, the host transfers the corresponding matrix to the global memory. Tiling parameters are then set to configure the kernel launch dimensions. The algorithm adapts the tile sizes (bm, bn) based on the query and database dimensions to exploit thread utilization and memory locality (Steps 3-4). The CUDA *grid* and *block* sizes are determined based on tile sizes to parallelize the workload over matrix dimensions (Steps 5).

Algorithm 4 Multi-request Parallel Goldberg-PIR

```
    $\underline{\boldsymbol{\rho} \leftarrow \texttt{PSD.Query.Response}(\boldsymbol{q})}$: Upon receiving multiple PIR
    requests $\boldsymbol{q} = \{q_i = (q_{i1}, q_{i2}, \ldots, q_{ir}) \in \mathbb{F}^r\}_{i=1}^{\bar{\boldsymbol{q}}}$
 1:     copy $\boldsymbol{q}$ from main memory to global memory
 2:     $br = 8$ , $tn = 8$ , $tm = 8$                              ▷ tiling setting
 3:     if $\bar{\boldsymbol{q}} \geq 128$ and $n \geq 128$ then $bm = 128$ , $bn = 128$
 4:     else $bm = 64$ , $bn = 64$
 5:     $|grid| = (\lfloor\frac{n}{bn}\rfloor, \lfloor\frac{m}{bm}\rfloor)$, $|block| = (bn, bm)$
 6:     CUDA Kernel:
 7:         for each block $(b_x, b_y) \in |grid|$ do
 8:             for each thread $(t_x, t_y) \in |block|$ do
 9:                 $r_x = tm \cdot b_x + t_x$ , $r_y = tn \cdot b_y + t_y$
10:                 $\rho_{c_x,c_y} = 0, \forall c_x \in \{1, \ldots tm\}, c_y \in \{1, \ldots, tn\}$
11:                 for $k = 0, \ldots, r$ [step $br$] do
12:                     copy $(\boldsymbol{q}_{r_x,[k,\ldots,k+br]}$ , $\texttt{DB}_{[k,\ldots,k+br],r_y})$ to shared
    memory
13:                     synchronize threads in the block $(b_x, b_y)$
14:                     $s \leftarrow \sum_{i=1}^{br} \boldsymbol{q}_{r_x,k+i} \cdot \texttt{DB}_{k+i,r_y} \bmod q$
15:                     $\rho_{t_y,t_x} \leftarrow \rho_{ty,t_x} + s \bmod q$
16:                     synchronize threads in the block $(b_x, b_y)$
17:                 copy $\rho_{t_y,t_x}$ to global memory
18:     return $\boldsymbol{\rho} = \{\rho_i = \{\rho_{i,1}, \ldots, \rho_{i,b}\}\}_{i=1}^{\bar{\boldsymbol{q}}}$
```

Within the CUDA kernel, each thread block computes a tile of the result matrix. Each thread performs a series of multiply-accumulate operations over the shared memory tiles to compute partial results for its assigned output element. These partial results are stored in thread-local registers and accumulated across all tile iterations (Steps 11-16). After full computation loop over the depth dimension r completes, each thread writes its final result to the output buffer ρ in global memory (step 17). Upon completion of the kernel execution, the host retrieves the response matrix $\boldsymbol{\rho}$ from the device memory and finally returns it as an output. The algorithm leverages several GPU optimization strategies inspired by best practices in high-performance matrix multiplication [43] such as shared memory caching, register blocking, warp-level parallelism, and conditional adjustment of tile sizes. These techniques collectively enhance arithmetic intensity and scalability, enabling practical deployment of PIR at scale.

E. Instantiation Optimizations

1) Offline-Online mode: The offline-online mode enhances efficiency across all phases of $\texttt{QPADL}$: *(i)* In the $\texttt{PoL}$ phase, static rings in the LRS scheme allow a single offline preprocessing step, ensuring stable performance across ring sizes. *(ii)* In the spectrum query phase, the main bottleneck which is the PIR responses that grow linearly with DB size, can be alleviated by offline-online precomputations on $\texttt{PSD}$ servers. *(iii)* In the service request phase, the $\texttt{HCT}$-based PoW can be converted to a non-interactive form via the Fiat-Shamir heuristic [44], enabling users to generate verifiable tokens independently and removing the need for interactive communication.

2) Database Compression Techniques: Since DB size directly affects $\texttt{PSD}$ computation, particularly in PIR, compression can substantially improve spectrum query performance in QPADL (see Section VI). QPADL employs a technique that sorts DB entries (e.g., FCC frequency data) and stores differences between successive items instead of full values [45]. As adjacent entries are often similar, their differences require fewer bits. This method applies to all block-based PIR schemes used in QPADL, reducing storage from $r_{\mathtt{DB}} \times b$ bits to about $O(r_{\mathtt{DB}}(b - \log(r_{\mathtt{DB}})))$ bits. The resulting efficiency gains in computation and storage are detailed in Section VI.

3) Multiple Block Retrieval: To improve efficiency, each DB index stores multiple puzzles of increasing difficulty, reducing the number of queries needed for different SAS services. We optimize communication by enabling multi-puzzle retrieval via PIR over PQ-Tor, with each PSD embedding several HCT or LBP puzzles into each block. This lets clients obtain multiple puzzles in a single query while preserving privacy and adding minimal overhead, improving system efficiency and reducing communication latency.

V. Security Analysis

We give a series of security proofs capturing the threat and security models as follows:

Theorem 1. *QPADL provides five key security guarantees: (i) λ-private, PQ-secure location privacy achieved through multi-server PIR; (ii) PQ computational anonymity via onion routing; (iii) PQ location verification through the unforgeability of LRSs; and (iv) PQ-DoS resilience by employing CPPs and rate-limiting mechanisms.*

Proof. (i) t-private PQ-secure Location Privacy: QPADL is instantiated with various PIR schemes, each ensuring at least t-private PQ secure location privacy, meaning that a coalition of up to $(t-1)$ PSDs learns nothing about the queried index θ (Definition 4). Both QPADL-ENS and QPADL-FTR offer IT security: $t = n-1$ in QPADL-ENS, and $0 \leq t \leq n-1$ in QPADL-FTR. Hence, their privacy guarantees remain unaffected by the adversary's computational power, including quantum capabilities [19]. Additionally, QPADL-OOP employs a PQ computationally secure PIR with $t = \pi$, where π denotes the number of item chunks in the DB, thus ensuring PQ location privacy during the spectrum query. The privacy guarantees of QPADL-ENS, QPADL-FTR, and QPADL-OOP are formally proven in Lemmas 1 and 2, respectively.

(ii) Client Anonymity and Untraceability: QPADL preserves client anonymity and untraceability against PSDs and eavesdroppers through onion routing with three relays (N_e, N_m, N_x), where each node knows only its predecessor and successor. Messages are transmitted over circuits layered with AES-256 encryption ($ctxt = \mathtt{Enc}_{sk_{N_1}}(\mathtt{Enc}_{sk_{N_2}}(\mathtt{Enc}_{sk_{N_3}}(m)))$, with keys $sk_{N_i} \in \{0,1\}^{256}$ derived via Module-LWE-based KEM encapsulation ($ctxt' \leftarrow \mathtt{ML\text{-}KEM.Encaps}(PK_{N_i,\mathtt{Kyber}}, sk_{N_i})$) and decapsulation ($sk_{N_i} \leftarrow \mathtt{ML\text{-}KEM.Decaps}(sk_{N_i,\mathtt{Kyber}}, ctxt')$) for $i = e, m, x$. Assuming IND-CPA security of the encryption and the hardness of solving worst-case Module-SIVP, any $\mathcal{A}$ observing the full circuit cannot link sender to communication with more than negligible probability (Definition 7). Overall security relies on AES-256's 128-bit PQ strength under Grover's algorithm [26] and Module-LWE, which reduces to worst-case MSIVP in the random oracle model (ROM) [46].

(iii) Location verification and Spoofing Resistance: In QPADL, location verification and spoofing resistance rely on the unforgeability and linkability of its PQ LRS scheme [24], built in the ROM. The scheme is instantiated via a hash-based non-interactive argument of knowledge (NIAoK), namely an augmented zero-knowledge variant of ethSTARK transformed into a SoK using the Fiat–Shamir heuristic. The signer's sk_l is committed as $PK_l := H'(sk_l)$, and a coalition of user keys is embedded in a Merkle tree to define the ring $\mathsf{R} = \{PK_1, \ldots, PK_n\}$, where $n = 2^k$ and $k = \lceil \log_2(n) \rceil$. The relation proven in the SoK is formalized as $\mathsf{R}_s = \{((e, rt, T), (P, l, sk_l)) : T = H'(sk_l, e),\ rt = \mathtt{MPath}(H'(sk_l), \mathbf{P}, l)t\}$, where $e \in \mathbb{F}_p$ is the event ID, T is the tag (enabling linkability), rt is the Merkle root, l is the binary index of PK_l in the tree, and $\mathbf{P}$ is the Merkle path. For any two signatures σ_1, σ_2 on messages m_1, m_2, linkability holds if their tags satisfy $H'(sk_l, e_1) = H'(sk_l, e_2)$. Thus, re-use of a location proof (e.g., from the same AP) is detectable. The zero-knowledge property of the NIAoK ensures no information beyond validity leaks, and the signature's unforgeability is reduced to the collision resistance and preimage resistance of H', along with the non-slanderability and extractability of the underlying SoK ([24] for more details). Under the hardness of the structured hash assumptions and the one-wayness of MPath, any adversary $\mathcal{A}$ attempting to forge a valid PoL or violate linkability has success probability bounded by $Adv^{\text{forge}}_{\mathcal{A}}(\lambda) \leq Adv^{\text{SoK}}_{\mathcal{A}'}(\lambda) + Adv^{\text{Hash}}_{\mathcal{A}''}(\lambda) \leq negl(\lambda)$, where $\mathcal{A}'$ breaks SoK extractability and $\mathcal{A}''$ finds a hash collision or preimage. Thus, QPADL ensures PoLs are both unforgeable and linkable, providing protection against spoofing attacks under standard assumptions (Definition 8).

(iv) Puzzle Authenticity and Counter-DoS Guarantees: In QPADL, puzzle authenticity is enforced through the existential unforgeability of the ML-DSA signature under chosen-message attacks (EUF-CMA). Let $(\Pi_\theta, \sigma_\theta)$ denote a valid puzzle-signature pair. For any PPT adversary $\mathcal{A}$, the advantage of forging a valid pair not issued by an authorized PSD is bounded as $Adv^{EUF\text{-}CMA}_{\mathcal{A}}(\lambda) := Pr[\mathcal{A}\ outputs\ (\Pi^*_\theta, \sigma^*_\theta) \notin \mathcal{Q}] \leq negl(\lambda)$, where $\mathcal{Q}$ is the set of honestly issued puzzles. This security is based on the hardness of the Module-LWE and Module-SIS challenges, with ML-DSA admitting a tight reduction to MSIS in the quantum ROM, guaranteeing authenticated puzzle issuance and preventing forgery.

The counter-DoS mechanism in QPADL for SAS servers is based on CPPs, relying on either the second preimage resistance of a hash function (QPADL-HCT) or the hardness of the Hermite-SVP problem (QPADL-LBP), as proven in Lemmas 3 and 4, respectively. To enforce per-client rate-limiting at PSDs, QPADL relies on the linkability of event-oriented LRS. Specifically, it embeds unique event IDs into the commitments of the form $h((l_x, l_y)||\beta_{\mathtt{TS}}||\mathtt{TS}||r)$, where each component is cryptographically bound, ensured by the one-wayness and collision resistance of the underlying H. $\mathcal{A}$ attempting to circumvent this limit must find a hash collision between commitments differing only in nonce, i.e., solve $h((l_x, l_y)||\beta_{\mathtt{TS}}||\mathtt{TS}||r) = h((l_x, l_y)||\beta_{\mathtt{TS}}||\mathtt{TS}||r')$, which succeeds with probability at most $2^{|H|/2}$ effort by the birthday bound for a hash of length $|H|$. Thus, the number of valid queries is bounded by the number of distinct APs (or ring members) in the region. Therefore, QPADL guarantees strong puzzle authenticity, per-request PoW hardness, and enforced rate limits. □

Lemma 1. *`QPADL-ENS` offers $\ell-1$ IT-secure location privacy in block retrieval relying on $\ell \geq 2$ non-colluding servers, while `QPADL-FTR` ensures t-private, ν-Byzantine robustness, k-out-of-ℓ IT-secure location privacy.*

Proof. In `QPADL-ENS`, the client selects r-bit binary strings $\{\rho_i\}_{i=1}^{\ell-1} \in \mathbb{GF}(2)^r$ uniformly at random and sets $\rho_\ell := \bigoplus_{i=1}^{\ell-1} \rho_i \oplus e_\theta$, with e_θ being the unit vector at position θ. The final response is $D_\theta := \bigoplus_{i=1}^{\ell} \rho_i$. For any coalition of corrupted servers $C \subset \{1,\ldots,\ell\}$ with $|C| \leq \ell-1$, and for any pair of indices $\theta', \theta'' \in [r]$, it holds that $\Pr[\{\rho_i\}_{i\in C} \mid \theta = \theta'] = \Pr[\{\rho_i\}_{i\in C} \mid \theta = \theta'']$, implying zero distinguishing advantage between queries. This shows that the distribution of queries received by each `PSD` is independent of θ. Assuming all servers respond honestly, `QPADL-ENS` ensures perfect (IT) privacy during the query phase. In `QPADL-FTR`, the client encodes θ using r random degree-t polynomials $\{f_j(x)\}$ over $\mathbb{F}[x]$ such that $f_j(0) = e_\theta[j]$, and sends to the $\texttt{PSD}_j$ the query $\rho_j := \langle f_1(\alpha_j), \ldots, f_r(\alpha_j)\rangle$, receiving response $R_j := \rho_j \cdot \texttt{DB}$. For any coalition $C \subset [\ell]$ with $|C| \leq t$, and any $\theta', \theta'' \in [r]$, it holds that $\Pr[\{\rho_i\}_{i\in C} \mid \theta = \theta'] = \Pr[\{\rho_i\}_{i\in C} \mid \theta = \theta'']$, yielding unconditional privacy for the target index θ and zero advantage for any unbounded $\mathcal{A}$. Moreover, `QPADL-FTR` ensures block reconstruction despite failures or malicious servers by using the Guruswami-Sudan list decoding algorithm, which corrects up to $\nu < k - \lfloor\sqrt{kt}\rfloor$ Byzantine responses when $k > t$ servers reply under (ℓ, t)-Shamir secret sharing. □

Lemma 2. *`QPADL-OOP` ensures π-private computationally-secure location privacy relying on the security of PRG.*

Proof. Let the DB be identically replicated across ℓ servers, with each block split into π chunks $\{x^{(j)}\}_{j=1}^{\pi}$, and with chunk order differing across servers. During preprocessing, SU selects a random λ-bit seed and applies a secure PRG to derive π query vectors. Responses are computed as $R_i := A_i \oplus q_i \cdot \texttt{chunk}_0$, where $A := q \cdot \texttt{DB}$ and $q := \texttt{PRG}(S_i, k(t-1))$, as shown in Fig. 1. Assuming a secure PRG, the `QPADL-OOP` achieves PQ-secure computational π-private location privacy against any coalition of $C \subset [\ell]$ with $|C| < \ell$. Since the PRG outputs are indistinguishable from uniform and the flip chunk is hidden from C, the $\mathcal{A}$'s view $\{{\rho_i}^{(j)}\}_{i\in C, j\in[\pi]}$ is computationally indistinguishable for any pair $\theta', \theta'' \in [r]$, satisfying: $|\Pr[\mathcal{A}(\{{\rho_i}^{(j)}\}_{i\in C}) \mid \theta = \theta'] - \Pr[\mathcal{A}(\{{\rho_i}^{(j)}\}_{i\in C}) \mid \theta = \theta'']| \leq negl(\lambda)$, where the distinguishing advantage is bounded as: $Adv_{\mathcal{A}}^{\text{Privacy}}(\lambda) := \max_{\theta',\theta''} |\Pr[\mathcal{A}(view_{\theta'})] - \Pr[\mathcal{A}(view_{\theta''})]| \leq Adv_{\mathcal{A}'}^{\text{PRG}}(\lambda)$ with $\mathcal{A}'$ being a reduction that breaks the PRG's PQ security. □

Lemma 3. *`QPADL-HCT` provides parallelized-resistant DoS protection for SAS servers using client-server puzzles based on the pre-image resistance of cryptographic hash functions.*

Proof. In `QPADL-HCT`, the client solves a sequence of $\ell = \lceil \log_2(n_l) \rceil$ hashcash puzzles $\pi = (h, n_s, \kappa, n_l)$ along a randomly selected path in a binary hash tree with n_l leaves. For each puzzle, the adversary requires $O(2^{|H|})$ trials with $|H|$ denoting the hash size and the success probability per attempt is $2^{-\kappa}$, requiring an average of $O(2^\kappa)$ hash in classical settings and $O(2^{\kappa/2})$ under Grover's algorithm. During randomized path-bound verification, the client must present valid solutions for all puzzles along the selected path, having committed to a valid root. This yields a total expected work of $\lceil \log_2(n_l) \rceil \cdot 2^\kappa$ per request, and $\mathcal{A}$'s success probability is bounded by $\log_2(n_l) \cdot 2^{-\kappa}$, which is negligible in κ. Since puzzles at higher levels depend on solving those below, at least $\lceil \log_2(n_l) \rceil$ sequential PoW are enforced, making the scheme resistant to full parallelization. The cost of each request grows exponentially with κ and linearly with the tree height, imposing a substantial burden on $\mathcal{A}$'s attempting large-scale flooding. Thus, `QPADL-HCT` achieves κ-bit DoS resistance with $\mathcal{A}$'s success probability bounded by $negl(\kappa)$, assuming the second-preimage resistance of $\mathcal{H}$. □

Lemma 4. *`QPADL-LBP` mitigates DoS attacks on SAS servers using client-server puzzles based on the hardness of the Hermite Shortest Vector Problem (Hermite-SVP).*

Proof. In `QPADL-LBP`, lattice-based PoW $\pi = (\alpha, n_\Lambda, B, p)$ relies on the hardness of the α-Hermite Shortest Vector Problem. Specifically, the challenge is to compute a nonzero vector of the lattice $\mathbf{v} \in \Lambda(B)\backslash\{0\}$ such that $||\mathbf{v}|| \leq \alpha \cdot \lambda_1(\Lambda)$, where $\lambda_1(\Lambda) = min\{||u|| | u \in \Lambda\backslash\{0\}\} \leq p^{1/n_\Lambda}$ is the length of the shortest nonzero lattice vector and α is a tunable approximation factor set to $\alpha = 1.05 \cdot \Gamma(n_\Lambda/2+1)^{1/n_\Lambda}/\sqrt{\pi}$ with n_Λ as the lattice dimension. Solving these puzzles employing the best-known techniques, such as lattice reduction and enumeration, requires on average $O(2^{0.2925 \times n_\Lambda + o(n_\Lambda)})$ trials in classical settings and $O(2^{0.2570 \times n_\Lambda + o(n_\Lambda)})$ trials in quantum settings to obtain a valid token for the servers. Thus, the success probability of $\mathcal{A}$ satisfies $Pr[\mathcal{A}\ finds\ \mathbf{v} \in \Lambda(B)\ s.t.\ ||\mathbf{v}|| \leq \alpha \cdot \lambda_1(\Lambda(B))] \leq 2^{-\kappa}$ which is negilgible in κ. Each puzzle remains valid for a limited duration, determined by κ, which depends on α, λ, and n_Λ. □

VI. Performance Evaluation and Comparison

This section outlines our evaluation metrics, and implementation setup, followed by a comprehensive assessment of `QPADL` across multiple instantiations, using diverse cryptographic techniques, optimizations, and GPU acceleration.

A. Configuration and Experimental Setup

Hardware: We evaluated the efficiency of `QPADL` framework using a standard desktop equipped with an Intel Core *i9-11900K*@3.50 *GHz*, 64 GiB RAM, 1 TB SSD, running Ubuntu 22.04.4 LTS. It is also equipped with NVIDIA GTX 3060 GPU card, which provides CUDA 3584 cores, 12GB GDDR6-based memory, and 360GB/s memory bandwidth.

Libraries: We used *C* and *Python* programming languages along with several cryptographic libraries, including: *percy++*[1] for multi-server PIR components, *liboqs*[2] for PQ-secure primitives, *OpenSSL* for standard cryptographic operations such as hash functions, and *NTL* for lattice-based puzzles. We have also used the LRS repository[3] for the ring signature and the hashcash-tree repository[4] for the hash-based puzzle. Additionally, DBs were constructed using *SQLite*[5]. For CPU-bounded implementation, we consider AVX instructions and OpenMP [47] as in [33].

[1]https://percy.sourceforge.net/

[2]https://openquantumsafe.org/

[3]https://github.com/yuxi16/Post-Quantum-Linkable-Ring-Signature?tab=

[4]https://github.com/alviano/hashcash-tree

[5]The DB is modeled as a matrix with adjustable row counts (e.g., 2^{10}, 2^{12}, 2^{14}, 2^{16}). For each grid cell (l_x, l_y), we generated synthetic spectrum records and embedded signed `HCT` and `LBP` puzzles, storing them across `PSD`s synchronized per FCC requirements [2].

Phase	Entity	Analytical Computational Cost	Communication Cost	Empirical Computational Cost (ms)											
PoL QPADL	User	$O(n_l) \cdot H' + O(\log\log(n_l)) \cdot H'$	$O(\text{polylog}(\log(n_{\texttt{AP}})))$	33.34											
	AP	$O(n_l) \cdot H' + O(\log(n_l) \cdot \log\log(n_l)) \cdot H'$		49.37											
Spectrum Query				$\lvert\texttt{DB}\rvert = 2^{12}$			$\lvert\texttt{DB}\rvert = 2^{14}$			$\lvert\texttt{DB}\rvert = 2^{16}$			$\lvert\texttt{DB}\rvert = 2^{18}$		
				$\bar{q} = 1$	$\bar{q} = 2^7$	$\bar{q} = 2^{10}$	$\bar{q} = 1$	$\bar{q} = 2^7$	$\bar{q} = 2^{10}$	$\bar{q} = 1$	$\bar{q} = 2^7$	$\bar{q} = 2^{10}$	$\bar{q} = 1$	$\bar{q} = 2^7$	$\bar{q} = 2^{10}$
Puzzle.Bind ($\kappa = 20$)	PSD-LBP (CPU)	$16r \cdot O(n_\sigma \log n_\sigma + 4n_\sigma) + r \cdot n_\Lambda^3 \cdot \text{mult}(n_\Lambda)$	$r \cdot \lvert\Pi\rvert$	11816			23320			69820			243602		
	PSD-LBP (GPU)			430			1470			5590			22248.2		
	PSD-HCT (CPU)			344			1376			5500			22625.61		
	PSD-HCT (GPU)			346			1370			5470			21825.33		
QPADL-ENS	User	$(r + b) \cdot ((\ell - 1) \cdot t_\oplus) + n \cdot t_\oplus$	$(r + b) \cdot \ell + O(\text{polylog}(\log(n_l)))$	0.258			0.260			0.273			0.294		
	PSD (CPU)			8.99	11.54	30.03	10.46	33.59	245.93	17.24	176.88	1334.0	42.86	681.42	5309.90
	PSD (GPU)			8.82	9.82	22.89	9.11	16.63	81.59	10.38	45.21	323.85	15.42	154.74	1269.30
QPADL-FTR	User	$\ell(\ell-1)rt_\oplus + 3\ell(\ell+1)t_\oplus + (n/w) \cdot t_\oplus$	$r \cdot w \cdot \ell + k \cdot b + O(\text{polylog}(\log(n_l)))$	3.62			6.31			7.51			8.65		
	PSD (CPU)			38.9	170.50	1058.9	133.7	821.3	5435.85	507.8	3087.5	19878.41	4106.94	16785.87	111319.09
	PSD (GPU)			22.18	37.60	149.79	55.41	102.77	574.64	199.22	389.76	2144.63	840.70	1720.35	9672.28
QPADL-OOP	User	$\sqrt{n} \cdot (r \cdot \ell + 1 + 1/\ell) \cdot t_\oplus + (n/2\ell)(1 + r - 1) \cdot t_\oplus$	$\ell\left(2\sqrt{n/(8\ell)} + \kappa/8\right) + O(\text{polylog}(\log(n_l)))$	2.02			2.72			5.30			9.43		
	PSD (CPU)			8.95	10.52	23.84	9.08	13.68	42.81	9.93	21.01	113.41	22.71	58.41	369.22
	PSD (GPU)			8.88	9.91	21.40	8.98	11.39	35.14	9.28	23.80	110.84	13.22	30.63	218.12
Service Request				$\kappa = 14$			$\kappa = 18$			$\kappa = 20$			$\kappa = 23$		
QPADL-HCT ($n_l = 2$)	User	$\lceil \log_2(n_l) \rceil \cdot 2^\kappa$	$\lambda + 32 + \lceil \log_2(n_l) \rceil \cdot \lvert H\rvert + O(\text{polylog}(\log(n_l)))$	38.94			66.36			316.56			6251.37		
	Server	$\lceil \log_2(n_l) \rceil \cdot H + O(n_\sigma \log n_\sigma + 4n_\sigma) + O(\log(n_l) \cdot \log\log(n_l)) \cdot H'$		10.53											
QPADL-LBP	User	$O(2^{0.2925 n_\Lambda + o(n_\Lambda)})$	$10n_\Lambda^2 + n_\Lambda(n_\Lambda - 1) + 10n_\Lambda^2 + n_\Lambda \log_2(\lVert\nu\rVert) + O(\text{polylog}(\log(n_l)))$	133.08			259.15			881.41			2931.45		
	Server	$O(n_\Lambda^2 \cdot \text{mult}(n_\Lambda)) + O(n_\sigma \log n_\sigma + 4n_\sigma) + O(\log(n_l) \cdot \log\log(n_l)) \cdot H'$		9.331			10.622			11.304			11.478		

All computation costs are reported in ms, and all communication costs are in Bytes. We set classical security at 128 bits per NIST guidelines and PQ security to NIST Level I, equivalent to 128-bit classical strength [7], with all parameters aligned accordingly. HCT uses SHA-256, and the LRS employs the Rescue-Prime hash function [48] with a SoK based on ethSTARK [25]. Here, ℓ is the number of responsive PSDs, w the number of words in the DB, b the size of each DB item in bits, r the number of rows, and $n = r \times b$ the total DB size in bits. n_l denotes the number of leaves of HCT, $n_{\texttt{AP}}$ the number of APs in the region, n_Λ the lattice dimension, and n_σ the lattice dimension in ML-DSA. $||\nu||$ represents the Euclidean norm of a lattice vector. $t_\oplus$ is the cost of an XOR operation, H denotes the cost of a hash operation, and H' the cost of the Rescue-Prime hash function. $mult(n_\Lambda)$ refers to multiplying two n_Λ-bit numbers. Finally, κ indicates the puzzle difficulty level in the quantum setting.

TABLE I: Analytical computational costs, communication overhead, and empirical (CPU/GPU) performance of QPADL.

Evaluation Metrics and Rationale: We evaluate QPADL both analytically and empirically, measuring computational cost and communication overhead across all entities and spectrum access phases. Our assessment covers LRS costs in the PoL phase, PIR overhead in QPADL-ENS, QPADL-FTR, and QPADL-OOP under CPU and GPU implementations, PSD DB setup costs, and PoW costs in QPADL-HCT and QPADL-LBP, along with PQ-Tor communication overhead. We further analyze scalability in terms of end-to-end delay under varying user loads, network conditions, and PSD configurations. Since QPADL is the first framework to jointly achieve location privacy, anonymity, location verification, and counter-DoS with PQ security, direct comparisons are not feasible; instead, we provide a comprehensive standalone evaluation and contrast specific metrics with related works offering partial overlap.

B. Experimental Results

The analytical and empirical evaluation of cryptographic, computational, and communication overhead across each phase of QPADL is shown in TABLE I and elaborated below:

1) Computational Costs: (i) PoL Phase: On the user side, this phase involves generating a location commitment via hashing and verifying the LRS, while the AP performs LRS signing. Signing includes an offline Merkle tree construction and root computation, followed by an online phase with Merkle path computation, hashing, and statement authentication using a SoK signature. Verification mirrors the offline setup and adds SoK verification. AP signing scales efficiently with ring size, from 20 ms for 2^3 members to 60 ms for 2^{13}, making it suitable for large SASs. Verification is lightweight, 0.4 ms for 2^3 users and 8 ms for 2^6, and ProxVerif adds only 1–10 ms using signal strength and RTT. *(ii) Puzzle Binding and Database Setup:* This phase involves generating HCT and LBP puzzles, signing them with ML-DSA, and binding them to each DB entry across varying DB sizes. ML-DSA key generation, signing, and verification take approximately 29 μs, 84 μs, and 30 μs, respectively. Puzzle generation for HCT involves selecting a random string, while LBP requires generating a lattice basis using uniformly random numbers. The combined overhead of puzzle creation and signature operations across different DB sizes via CPU and GPU-parallelized implementation is detailed in TABLE I.

(iii) Spectrum Query Phase: This phase integrates three PIR schemes involving PIR.Query, PIR.Query.Response, and PIR.BlockReconst operations, along with LRS verification and linkability checks on the PSD side, all executed over PQ-Tor. PQ-Tor overhead includes circuit setup and layered encryption, primarily driven by three ML-KEM and AES operations. Specifically, ML-KEM key generation, encapsulation, and decapsulation take 10 μs, 13.4 μs, and 9 μs, respectively, while AES-256 requires 7 μs for key generation and 8 μs for encryption. *(iv) Service Request Phase:* This phase requires PoW on the client side using HCT or LBP, and on the server side, LRS verification and puzzle authentication via ML-DSA. Solving HCT requires approximately $\lceil \log_2(n_l) \rceil \cdot 2^\kappa$ hash operations, taking 38-316 ms for κ up to 18. In contrast, LBP employs lattice basis reduction and enumeration [49], with runtimes ranging from 133-881 ms. For larger κ, lattice-based puzzles help maintain practical solving times (3 instead of 6 s) over hash-based ones while mitigating DoS attacks.

2) Parallelization Assessment with GPU: Fig. 2 shows the runtime of QPADL-ENS, QPADL-FTR, and QPADL-OOP at the PSD side for varying DB sizes and query counts ($\bar{q}$), with each DB entry fixed at 3KB. GPU-accelerated implementations achieve up to an order-of-magnitude speedup over CPU-bound versions. Among them, QPADL-OOP has the lowest online server cost due to its offline-online design; for instance, with a 128MB DB and 2^{10} queries, it outperforms QPADL-ENS and QPADL-FTR by 3.2× and 16.5×, respec-

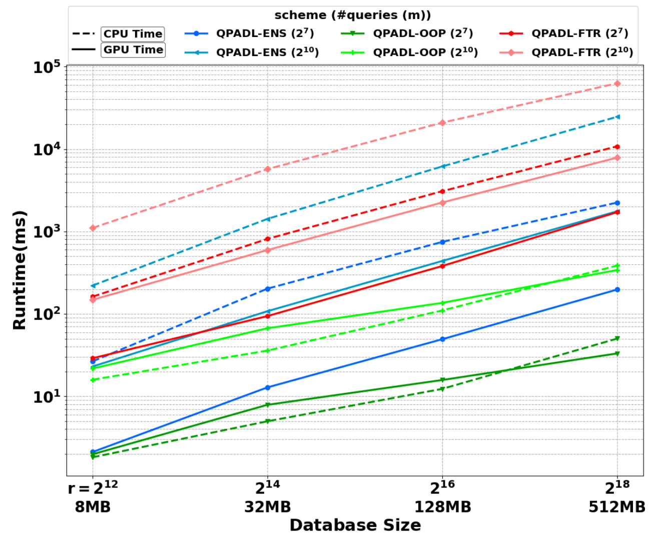


Fig. 2: Scalability Benchmarking of PIRs on CPU/GPU

tively. This benefit, however, comes with $\approx 19\times$ higher client computation, extra offline precomputation at each PSD, and an additional offline interaction with the user. We emphasize that GPU performance gains over CPU increase with larger database and query sizes, particularly in QPADL−OOP, which more accurately reflects real-world requirements. Additionally, GPU parallelization can enhance the performance of Puzzle.Bind. For example, GPU-based construction of databases containing LBP provides up to $10\times$ speedup on databases with 2^{18} entries. For HCT, CPUs benefit from highly optimized OpenSSL hash functions, outperforming a single GPU core. Nevertheless, the GPU's advantage over the CPU becomes more pronounced as the database size exceeds 512 MB. In GPU-accelerated PIR, keeping the static DB in GPU global memory avoids repeated CPU–GPU transfers, improving throughput. Our benchmarks demonstrate the advancement of GPU over CPU for multi-server PIRs and puzzle generation in Puzzle.Bind are available on GitHub [6].

3) Communication Overhead: *(i)* The location commitment includes the location coordinates (16 bytes), beacon (8 bytes), timestamp (8 bytes), and a random nonce (4 bytes). The ring signature size scales with the number of ring members, measuring approximately 18 KB for 2^3 users and around 20 KB for rings of size 2^6 to 2^{13}. *(ii)* In QPADL−ENS, QPADL−FTR, and QPADL−OOP, the communication complexity involves retrieving a b-bit block from ℓ responsive PSDs, along with transmitting the location commitment and PoL. Each b-bit block contains 560 bytes of spectrum data, an HCT or LBP puzzle, and the PSD's ML-DSA signature. The HCT puzzle includes a λ-bit nonce (n_s), a 4-byte difficulty (κ), and a 1-byte level (n_l), totaling 37 bytes. The LBP puzzle contains a $10n$-bit prime (p), $(n-1)$ samples of size $10n$ bits, plus 4-byte values for α and n_Λ, totaling 28133 bytes. Since spectrum queries run over PQ-Tor, its communication overhead directly impacts delays. PQ-Tor's performance closely matches conventional Tor, with only minor differences from ML-KEM and AES-256 operations. Thus, we utilized conventional Tor network metrics for communication delay estimation [50]. Although ML-KEM is faster than RSA (used in Tor), its use still requires two packet

[6]https://github.com/kiarashsedghigh/GPU-Multiserver-PIR-PuzzleGen.git

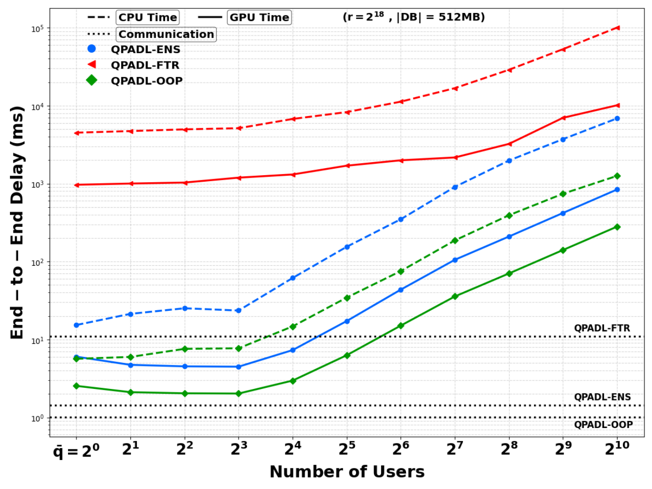


Fig. 3: End-to-End Cryptographic Delay

transmissions due to Tor's 512-byte packet size, resulting in an average circuit build time of about 300 ms. As each retrieved block remains under 50 KB, the PQ-Tor communication delay is bounded at approximately 175 ms. *(iii)* In service request phase, the client transmits the location commitment, PoL, and the signed token. The HCT solution size is $\lceil \log_2(n_l) \rceil \times |H|$ bits, while the LBP solution size is $n_\Lambda \times \lceil \log_2(2\alpha \cdot p^{1/n_\Lambda}) \rceil$ bits. The accompanying ML-DSA signature is 2420 bytes.

4) Scalability Assessment: We evaluate scalability through end-to-end (E2E) cryptographic delay, covering both computation and communication in spectrum access, including PoL acquisition, spectrum query, and puzzle retrieval. Fig. 3 shows that for few SUs, communication dominates, whereas with more SUs and large DB size, computation becomes the bottleneck. GPU acceleration overcomes this: QPADL−ENS achieves 2.66–4.18×, QPADL−FTR 4.83–11.49×, and QPADL−OOP 1.66–1.71× speedup for up to 2^{10} SUs per grid and time window. Although GPU-accelerated QPADL−OOP yields the best performance, its speedup is smaller due to lower CPU overhead, while the GPU advantage of QPADL−ENS and QPADL−FTR grows with larger query volumes and DB sizes.

5) Comparison with SOTA: Existing PIR-based schemes [2], [5], [6] protect location privacy but lack strong anonymity or PQ assurances [9], [10], [12], whereas QPADL enhances privacy in the query phase through optimizations and hardware acceleration while providing PQ security. Most SAS privacy solutions assume honest users, overlooking spoofing, and existing location verification approaches rely on group signatures, lack PQ guarantees, and incur high costs (over 100 ms) with trusted servers [5], [11], [12]. Our PoL achieves PQ security with sub-100 ms performance (33.34 ms on SUs and 49.37 ms on APs). Also, unlike prior works that focus only on DoS detection (e.g., AI-based methods), QPADL offers comprehensive counter-DoS for both PSDs and servers against quantum-capable adversaries [3], [13]–[15]. The efficiency of QPADL for real-world SAS deployment during the PQ transition is demonstrated in TABLE I.

VII. Conclusion

This work introduced QPADL, the first framework to simultaneously address location privacy, anonymity, location spoofing, and DoS threats in SASs under PQ security assumptions. By integrating privacy-preserving spectrum queries with robust CPPs, `QPADL` mitigates both conventional and PQ threats while maintaining scalability for large-scale SAS. Our proposed instantiations, built on diverse cryptographic primitives, offer flexible security-performance trade-offs. Formal security analysis confirms the framework's resilience, and extensive performance evaluations, enhanced through GPU parallelization and optimization, demonstrate its practicality and efficiency, establishing `QPADL` as a viable and future-proof solution for secure spectrum access.

Acknowledgment

This work is supported by the NSF-SNSF (2444615) and NSF CNS-2350213.

References

[1] M. Grissa, A. A. Yavuz, and B. Hamdaoui, "Preserving the location privacy of secondary users in cooperative spectrum sensing," *IEEE Transactions on Information Forensics and Security*, vol. 12, 2016.

[2] M. Grissa, A. A. Yavuz, B. Hamdaoui, and C. Tirupathi, "Anonymous dynamic spectrum access and sharing mechanisms for the cbrs band," *IEEE Access*, vol. 9, pp. 33 860–33 879, 2021.

[3] T. Chakraborty, S. Mitra, and S. Mittal, "Capow: Context-aware ai-assisted proof of work based ddos defense," *arXiv preprint*, 2023.

[4] D. K. Jasim and S. B. Sadkhan, "Cognitive radio network: Security and reliability trade-off-status, challenges, and future trend," in *2021 1st Babylon International Conference on Information Technology and Science (BICITS)*. IEEE, 2021, pp. 149–153.

[5] J. Xin, M. Li, C. Luo, and P. Li, "Privacy-preserving spectrum query with location proofs in database-driven crns," in *2016 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2016, pp. 1–6.

[6] S. Darzi and A. A. Yavuz, "Privacy-preserving and post-quantum counter denial of service framework for wireless networks," in *MILCOM 2024-IEEE Military Communications Conference*. IEEE, 2024.

[7] R. Bavdekar, E. J. Chopde, A. Agrawal, A. Bhatia, and K. Tiwari, "Post quantum cryptography: a review of techniques, challenges and standardizations," in *2023 International Conference on Information Networking (ICOIN)*. IEEE, 2023, pp. 146–151.

[8] P. Agarwal, M. Manekiya, T. Ahmad, A. Yadav, A. Kumar, M. Donelli, and S. T. Mishra, "A survey on citizens broadband radio service (cbrs)," *Electronics*, vol. 11, no. 23, p. 3985, 2022.

[9] R. Zhu, L. Xu, Y. Zeng, and X. Yi, "Lightweight privacy preservation for securing large-scale database-driven cognitive radio networks with location verification," *Security and Communication Networks*, 2019.

[10] M. Ul Hassan, M. H. Rehmani, M. Rehan, and J. Chen, "Differential privacy in cognitive radio networks: a comprehensive survey," *Cognitive Computation*, vol. 14, no. 2, pp. 475–510, 2022.

[11] N. Nguyen-Thanh, D.-T. Ta, and V.-T. Nguyen, "Spoofing attack and surveillance game in geo-location database driven spectrum sharing," *IET Communications*, vol. 13, no. 1, pp. 74–84, 2019.

[12] S. Darzi and A. A. Yavuz, "Slap: Secure location-proof and anonymous privacy-preserving spectrum access," *preprint arXiv:2503.02019*, 2025.

[13] G. Jakimoski and K. Subbalakshmi, "Denial-of-service attacks on dynamic spectrum access networks," in *IEEE International Conference on Communications Workshops*. IEEE, 2008, pp. 524–528.

[14] S. Darzi and A. A. Yavuz, "Counter denial of service for next-generation networks within the artificial intelligence and post-quantum era," in *2024 IEEE 6th International Conference on Trust, Privacy and Security in Intelligent Systems, and Applications (TPS-ISA)*. IEEE, 2024.

[15] R. Doriguzzi-Corin and D. Siracusa, "Flad: adaptive federated learning for ddos attack detection," *Computers & Security*, vol. 137, 2024.

[16] V. Bostanov, "Client puzzle protocols as countermeasure against automated threats to web applications," *IEEE Access*, vol. 9, 2021.

[17] I. M. Ali, M. Caprolu, and R. D. Pietro, "Foundations, properties, and security applications of puzzles: A survey," *ACM Computing Surveys (CSUR)*, vol. 53, no. 4, pp. 1–38, 2020.

[18] B. Chor, E. Kushilevitz, and O. Goldreich, "Private information retrieval," *Journal of the ACM (JACM)*, vol. 45, no. 6, pp. 965–981, 1998.

[19] I. Goldberg, "Improving the robustness of private information retrieval," in *2007 IEEE Symposium on Security and Privacy*. IEEE, 2007.

[20] A. Back *et al.*, "Hashcash-a denial of service counter-measure," 2002.

[21] T. Dang, J. Lichtinger, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, and R. Perlner, "Module-lattice-based digital signature standard," 2024.

[22] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, Z. Zhang *et al.*, "Falcon: Fast-fourier lattice-based compact signatures over ntru," *NIST's post-quantum cryptography standardization process*, vol. 36, no. 5, 2018.

[23] D. Cooper *et al.*, "Stateless hash-based digital signature standard," 2024.

[24] Y. Xue, X. Lu, M. H. Au, and C. Zhang, "Efficient linkable ring signatures: new framework and post-quantum instantiations," in *European Symposium on Research in Computer Security*. Springer, 2024.

[25] S. Team, "ethstark documentation–version 1.1," IACR preprint archive 2021, Tech. Rep., 2021.

[26] X. Bonnetain, M. Naya-Plasencia, and A. Schrottenloher, "Quantum security analysis of aes," *IACR Transactions on Symmetric Cryptology*, vol. 2019, no. 2, pp. 55–93, 2019.

[27] N. I. of Standards and Technology, "Fips 203 module-lattice-based key-encapsulation mechanism standard," 2024.

[28] Z. Gao, H. Zhu, Y. Liu, M. Li, and Z. Cao, "Location privacy leaking from spectrum utilization information in database-driven cognitive radio network," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 1025–1027.

[29] B. Bahrak, S. Bhattarai, A. Ullah, J.-M. J. Park, J. Reed, and D. Gurney, "Protecting the primary users' operational privacy in spectrum sharing," in *2014 IEEE International Symposium on Dynamic Spectrum Access Networks (DYSPAN)*. IEEE, 2014, pp. 236–247.

[30] L. Jing, L. Ke, Z. Lei, Y. Xiaoya, J. Yuanyuan, and J. Huinan, "Geohash coding location privacy protection scheme based on entropy weight topsis," *The Journal of Supercomputing*, vol. 81, no. 1, p. 85, 2025.

[31] M. Robinson and I. Psaromiligkos, "Received signal strength based location estimation of a wireless lan client," in *IEEE Wireless Communications and Networking Conference*, vol. 4. IEEE, 2005.

[32] J. Koo and H. Cha, "Localizing wifi access points using signal strength," *IEEE Communications letters*, vol. 15, no. 2, pp. 187–189, 2010.

[33] D. Günther, M. Heymann, B. Pinkas, and T. Schneider, "{GPU-accelerated}{PIR} with {Client-Independent} preprocessing for {Large-Scale} applications," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 1759–1776.

[34] T. Aura, P. Nikander, and J. Leiwo, "Dos-resistant authentication with client puzzles," in *International workshop on security protocols*. Springer, 2000, pp. 170–177.

[35] V. Chiriaco, A. Franzen, and R. Thayil, "Finding partial hash collisions by brute force parallel programming," in *2017 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*. IEEE, 2017.

[36] M. Alviano, "Hashcash tree, a data structure to mitigate denial-of-service attacks," *Algorithms*, vol. 16, no. 10, p. 462, 2023.

[37] R. Behnia, E. W. Postlethwaite, M. O. Ozmen, and A. A. Yavuz, "Lattice-based proof-of-work for post-quantum blockchains," in *International Workshop on Data Privacy Management*. Springer, 2021.

[38] "Darmstadt svp challenge," https://www.latticechallenge.org/svp-challenge/, 2024, accessed: April, 2024.

[39] M. Lam, J. Johnson, W. Xiong, K. Maeng, and U. Gupta, "Gpu-based private information retrieval for on-device machine learning inference," *arXiv preprint arXiv:2301.10904*, 2023.

[40] Q. Li and R. Zong, "Cat: A gpu-accelerated fhe framework with its application to high-precision private dataset query," *arXiv preprint arXiv:2503.22227*, 2025.

[41] D. Kirk *et al.*, "Nvidia cuda software and gpu parallel computing architecture," in *ISMM*, vol. 7, 2007, pp. 103–104.

[42] L. Zhang, J. Huang, S. Di, S. Matsuoka, and M. Wahib, "Can tensor cores benefit memory-bound kernels?(no!)," in *Proceedings of the 17th Workshop on General Purpose Processing Using GPU*, 2025, pp. 28–34.

[43] V. Kelefouras, A. Kritikakou, I. Mporas, and V. Kolonias, "A high-performance matrix–matrix multiplication methodology for cpu and gpu architectures," *The Journal of supercomputing*, vol. 72, no. 3, 2016.

[44] R. Canetti, Y. Chen, J. Holmgren, A. Lombardi, G. N. Rothblum, R. D. Rothblum, and D. Wichs, "Fiat-shamir: from practice to theory," in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, 2019, pp. 1082–1090.

[45] S. Tamrakar, J. Liu, A. Paverd, J.-E. Ekberg, B. Pinkas, and N. Asokan, "The circle game: Scalable private membership test using trusted hardware," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 2017, pp. 31–44.

[46] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, and V. Lyubashevsky, "Crystals-kyber: a cca-secure module-lattice-based kem," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2018.

[47] R. Chandra, *Parallel programming in OpenMP*. Morgan kaufmann, 2001.

[48] A. Szepieniec, T. Ashur, and S. Dhooghe, "Rescue-prime: a standard specification (sok)," *Cryptology ePrint Archive*, 2020.

[49] M. R. Albrecht, L. Ducas, G. Herold, E. Kirshanova, and E. W. Postlethwaite, "The general sieve kernel and new records in lattice reduction," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2019.

[50] "Tor metrics," https://metrics.torproject.org/torperf.html, accessed: 2024.