

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

Saleh Darzi^a, Kiarash Sedghighadikolaei^a, Saif Eddine Nouma^a and Attila Altay Yavuz^a

^aBellini College of Artificial Intelligence, Cybersecurity and Computing, University of South Florida, Tampa, 33620, Florida, USA

ARTICLE INFO

Keywords:

Privacy and Anonymity
Post-Quantum Security
Location Proof
Spectrum Access Systems
Counter DoS

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 advances in quantum computing. Thus, we propose QPADL, the first post-quantum (PQ) secure framework that simultaneously ensures privacy, anonymity, proximity-backed location assurance, 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 event-oriented linkable ring signatures for proximity-backed location verification alongside client puzzle protocols and an event-scoped 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.

1. Introduction

The rapid growth of wireless technologies (e.g., mobile and IoT) combined with regulated spectrum allocation (e.g., FCC in the U.S. [32]) has led to spectrum scarcity. Cognitive Radio Networks (CRNs) mitigate this challenge by enabling Secondary Users (SUs) to opportunistically access unused licensed channels [33]. At the core of this model lies the Spectrum Access System (SAS), which dynamically allocates spectrum through geo-location databases (DBs) [15]. Despite its effectiveness, SAS poses major privacy and security risks: continuous reporting of locations and transmission details compromises user privacy and anonymity [36], while its location-centric design enables spoofing, falsified data, and unauthorized access [58]. 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 [23]. These challenges are intensified by quantum computing, which threatens classical cryptographic protections [8]. *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.*

1.1. Related Works and Open Problems

(i) **Location Privacy and Anonymity in SAS:** Centralized SAS, mandated by the FCC, operates through multiple geo-location databases to facilitate dynamic spectrum sharing among governmental, commercial, licensed and unlicensed users [32]. 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 [36, 1]. These risks are especially acute for vehicle-mounted Citizens Broadband Radio Service Devices (CBSDs), an increasingly prominent class of mobile Citizens Broadband Radio Service (CBRS) adopters in transit, utility, and public-safety fleets, whose repeated location reporting exposes operational routes and trajectories. However, existing approaches remain inadequate: (1) k-anonymity and pseudo-identifier methods [61] lack provable security and provide only weak guarantees unless large anonymity sets are used, which is infeasible in large-scale SAS deployments; (2) differential privacy-based techniques [57], while theoretically sound, significantly degrade the accuracy of spectrum availability information; and (3) Private Information Retrieval (PIR)-based schemes [33, 58] 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 [23].

(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 [44]. Existing location verification schemes often assume trusted infrastructure, honest participants, or dedicated location servers, assumptions that are unrealistic in rural or infrastructure-limited settings [58]. Furthermore, most approaches fail to preserve location privacy and anonymity from the verifying entities themselves. These limitations underscore the need for a practical and

✉ salehdarzi@usf.edu (S. Darzi); kiarashs@usf.edu (K. Sedghighadikolaei); saifeddinouma@usf.edu (S.E. Nouma); attilaayavuz@usf.edu (A.A. Yavuz)
ORCID(s):

privacy-preserving location assurance framework capable of operating under real-world constraints while resisting diverse location-based attacks [24].

(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 [35]. 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) [22]. Machine learning has further advanced IDSs, enabling detection of abnormal traffic patterns with reported accuracies above 95% [15]. 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. In addition, AI-driven defenses remain susceptible to adversarial manipulation, where attackers exploit model weaknesses to evade detection, potentially incurring significant operational costs [25]. CPPs mitigate malicious requests by requiring clients to solve puzzles before service [12], but at scale, they suffer from bottlenecks in puzzle generation, distribution, and parallelization, imposing high costs on servers and legitimate users [3]. Outsourcing puzzle generation, as in [23], 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 [8]. 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 [23]. Ensuring durable protection requires adopting Post-Quantum Cryptography (PQC) to preserve privacy, anonymity, and availability against quantum-capable adversaries.

These four challenges cannot be solved by composing independent mechanisms, because in SAS they act on shared state and interfere when layered naively. A single spectrum query must conceal the user's location index (privacy), traverse a network that could deanonymize its sender (anonymity), originate from a user who may falsify that location (verification), and reach a database that adversaries can flood (DoS). The couplings are bidirectional: PIR hides the queried index but still leaks identity at the transport layer without anonymity; location verification certifies coordinates yet destroys privacy if the verifier learns them; and DoS defense needs the per-client linkability that anonymity

removes. This last tension, strong anonymity against per-client abuse control, is the hardest, since SAS demands both at once [24, 23]. An adequate solution must therefore satisfy all four jointly, within one query-and-service workflow, rather than as separable modules. No existing scheme does so, motivating the unified design we present next.

1.2. 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 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.

- **Proximity-backed Location Assurance 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 the spectrum query phase to verify SU locations and issue time-sensitive, quantum-safe proofs using signal strength for proximity checks. To 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 one-use-per-window admission at the PSDs. QPADL mitigates relay-based location spoofing such as distance hijacking through RTT-weighted proximity verification [44], 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 DoS resistance at both the query and service layers with regulations compliance.

- **Enabling Scalability Through Parallelization and Optimization:** We conducted comprehensive analytical and

Table 1

Qualitative comparison of QPADL with representative privacy-preserving SAS schemes.

Scheme	Setting	Security Assumption	PQ Ready	Location Privacy Mechanism	Anonymity Mechanism	Location Verification	DoS Resilience Mechanism	Adversary Model		Optimizations & Acceleration
								SU	DB	
Salami et al. [49]	1-DB	DCRA, Factoring	×	Paillier HE	×	×	×	Honest	HbC	Offline Precomputation (Optional)
Troja et al. [55]	1-DB	DDH, DL	×	PIR + AV-Net	AV-Net	×	×	Honest	HbC	P2P Cache Sharing P2P Query Reduction
Li et al. [43]	1-DB	DDH, DL	×	ElGamal + PPT	Pseudonym	AP + PPT	×	Malicious (location)	HbC	Offline Precomputation
Xin et al. [58]	1-DB	QRA, DL	×	QR-based PIR, Query Hiding	Ring Sig.	AP + Ring Sig.	×	Malicious (location)	HbC	Offline Precomputation
Grissa et al. [31]	ℓ-DB	IT	Partial	Multi-Server PIR	×	×	×	Honest	HbC, colluding + Byzantine	DB Partitioning, Batching, 2PRG-Compacted Queries
Grissa et al. [33]	ℓ-DB	IT, DDH, Strong RSA	Partial	Multi-Server PIR	EPID-AC	Spectrum-Misuse Detection	×	Byzantine	HbC, colluding	Query Batching, Blockchain, Clustering/Leader-Based Querying
SLAP [24]	1-DB	DDH, q-SDH, RSA	×	DAC + ZKPoK	DAC	AP + Group Sig. (+ DBP Fallback)	RSA-Based TLP	Malicious (location)	HbC	Online-Offline, Puzzle Precomputation, Batch Verification
PACDoSQ [23]	ℓ-DB	IT, M-LWE	✓	Multi-Server PIR	PQ-Tor	×	Hash-based CPP	Malicious (DoS)	HbC + Byzantine	Online-Offline, Puzzle Precomputation
QPADL	ℓ-DB	IT, M-LWE, M-SIS	✓	Multi-Server PIR (3 Variants)	PQ-Tor	AP + PQ-LRS	HCT CPP LBP CPP Rate-Limit (LRS)	Malicious (location) (DoS)	HbC Colluding + Byzantine	GPU Parallelization, Online-Offline, DB Compression, Multi-Block Retrieval Puzzle Precomputation, Batch Verification

Abbreviations: IT: Information-Theoretic; DL: Discrete Logarithm; DDH: Decisional Diffie-Hellman; DCRA: Decisional Composite Residuosity Assumption; QRA: Quadratic Residuosity Assumption; q-SDH: q -Strong Diffie-Hellman; M-LWE/M-SIS: Module Learning With Errors / Short Integer Solution; HE: Homomorphic Encryption; PIR: Private Information Retrieval; AV-Net: Anonymous Veto Network; PPT: Private Proximity Testing; DAC: Delegatable Anonymous Credential; ZKPoK: Zero-Knowledge Proof of Knowledge; EPID-AC: Enhanced Privacy ID Anonymous Credential; AP: Access Point; DBP: Distance-Bounding Protocol; CPP: Client Puzzle Protocol; HCT: Hash-Cash Puzzle; LBP: Lattice-Based Puzzle; TLP: Time-Lock Puzzle; LRS: Linkable Ring Signature; PQ-Tor: Post-Quantum Onion Routing; HbC: Honest-but-Curious; SU: Secondary User; DB: Database. **Symbols:** ✓: supported / post-quantum ready; ×: not supported; *Partial*: only part of the scheme is post-quantum secure. **Note:** Computational and communication cost comparisons are provided in Section 7.

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.

Positioning of QPADL. Table 1 contrasts the cryptographic components underlying QPADL with those of existing SAS privacy solutions. Beyond the individual mechanisms, QPADL is, to our knowledge, the first PQ framework to achieve location privacy, anonymity, location verification, and DoS resilience simultaneously while remaining compliant with FCC-mandated SAS architecture, and it does so under a more realistic system model than the state of the art: rather than assuming honest users or trusted servers, we permit compromised or malicious SUs that may falsify their location, model PSDs and SAS servers as honest-but-curious, and place trust only in the spectrum regulator.

Central to this is our use of event-oriented LRS as the proof-of-location artifact, which is PQ-secure and efficient, conceals AP identity and client coordinates from PSDs and servers, and through its event-scoped linkability, simultaneously enables a rate-limiting mechanism that protects the PSDs from SU-launched abuse, a threat no prior SAS scheme addresses. Our counter-DoS design further advances existing client-puzzle solutions by outsourcing puzzle generation and distribution to already-deployed entities, i.e., PSDs, in offline mode, and by offering two instantiations: a parallelization-resistant hash-based puzzle enforcing sequential client-side work, and a lattice-based

puzzle with a quantum security edge at the cost of higher communication. On the query side, multiple PIR instantiations let deployments select the security-efficiency trade-off matching their SAS configuration. Finally, GPU acceleration, database compression, offline-online execution, and multi-block retrieval improve efficiency beyond prior work, in several cases surpassing even classically secure schemes.

1.3. Improvements Over The PACDoSQ Paper[23]

This work is an extended version of *IEEE MILCOM-2024 Conference* [23], with 100% more new material and the following significant improvements and features:

(i) **Location Verification Mechanism:** We propose a PQ-secure location verification mechanism that uses LRSs and nearby WiFi APs to resist spoofing, in realistic networks with malicious or compromised SUs.

(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 instantiations: QPADL-ENS for efficiency, QPADL-FTR for fault-tolerant robustness, and QPADL-OOP for scalability with lower communications costs.

(iv) **Hardware Acceleration and Optimizations** We addressed the main bottleneck of [23] (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 subroutines, and proposed compression techniques.

2. 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 $(x_1, x_2, \dots, x_\ell)$. The notation $x \xleftarrow{\$} S$ indicates uniform random selection from the set S , and $\mathbf{v}$ denotes a vector. $||\cdot||$ 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 ID refer to the signature, ciphertext, and user identity, respectively.

2.1. System Architecture

Our system model consists of the following main entities within a regulated SAS-oriented spectrum-sharing framework, where unlicensed users request transmission authorization from certified geolocation databases based on their operational parameters [45]. This setting is consistent with practical deployments such as U.S. TV White Space (TVWS) and Citizens Broadband Radio Service (CBRS) in the 3.5 GHz band, in which spectrum-access decisions are mediated by authorized database administrators under regulatory oversight [13].

- **Spectrum Regulator:** The spectrum regulator (e.g., FCC) is the governing authority of the SAS. It defines and enforces operational rules, including incumbent protection, registration and reporting requirements, and transmission constraints, while certifying and auditing database operators and providing the regulatory parameters needed for spectrum-access decisions.
- **Servers:** Servers denote application- or cloud-layer service providers (e.g., CRN platforms, web services, edge or cloud servers) that users access after obtaining spectrum authorization. They provide network services but are not involved in spectrum query and allocation.
- **Private Spectrum Databases (PSDs):** PSDs are regulator-certified third-party geolocation DB operators (e.g., Google, Spectrum Bridge) that maintain incumbent records, protection data, and spectrum-access policies [1, 17]. Acting as the core decision entities in the SAS, they process client queries against regulatory and spectrum-usage data and return channel availability, transmission parameters, and validity intervals, while synchronizing regularly to preserve consistency under regulatory requirements [33].
- **Clients:** Clients are unlicensed wireless devices (e.g., mobile phones, laptops, IoT nodes, fixed terminals, or vehicle-mounted CBSDs such as transit, utility, and public-safety fleet units) that seek network access by querying PSDs for spectrum availability. After receiving authorization, they may opportunistically use licensed spectrum under the assigned operating constraints so as to avoid harmful interference to primary users. In this paper, the terms *client*, *user*, and *SU* are used interchangeably.

- **Access Points (APs):** AP denotes the set of wireless infrastructure nodes in the region, such as WiFi access points, LTE eNodeBs, and 5G gNodeBs, that provide last-hop connectivity for clients. These APs enable communication with PSDs and servers through the network backhaul and, in our setting, may additionally support synchronized operation for proximity-backed location assurance while remaining separate from spectrum decisions [1].

2.2. Cryptographic Building Blocks

(1) Private Information Retrieval: PIR enables a client to retrieve a data block from a database DB without revealing its index [19]. 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 [30].

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

- $\{q_i\}_{i=1}^n \leftarrow \text{Client.Query}(\theta, n)$: The client generates n queries for index θ and sends each to database server DB_i .
- $\rho_i \leftarrow \text{DB.Query.Response}(q_i)$: Each database server processes its query and returns a response ρ_i derived from its local database.
- $D_\theta \leftarrow \text{Client.BlockReconst}(\{\rho_i\}_{i=1}^n)$: The client reconstructs D_θ from the ℓ responses received from the database servers.

(2) Proof of Work: A PoW is a cryptographic mechanism based on puzzles with tunable difficulty [6] that are computationally easy to generate but hard to solve.

Definition 2. A PoW scheme operate as follows:

- $\Pi \leftarrow \text{Puzzle.Gen}(1^\lambda, \kappa)$: Given the security parameter λ and difficulty level κ , the algorithm generates a puzzle instance Π .
- $\Psi \leftarrow \text{PoW}(\Pi, \kappa)$: Given a puzzle Π and difficulty level κ , this algorithm computes a valid solution Ψ .
- $\{0,1\} \leftarrow \text{PoW.Verify}(\Pi, \Psi)$: 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 signatures under its PQC initiative [8]: ML-DSA (FIPS 204 [21]), FN-DSA [27], and SLH-DSA (FIPS 205 [20]). In QPADL, we adopt ML-DSA (Module LWE/SIS), with three algorithms: $(sk, PK) \leftarrow \text{ML-DSA.KeyGen}(1^\lambda)$, $\sigma \leftarrow \text{ML-DSA.Sign}(sk, m)$, and $\{0,1\} \leftarrow \text{ML-DSA.Verify}(PK, m, \sigma)$.

Ring signatures allow a user to anonymously sign a message on behalf of a group, while linkable ring signatures (LRS) further enable the detection of multiple signatures generated by the same signer within a given context. In event-oriented LRS, an event identifier is embedded in each signature, enabling linkability across signatures produced with the same key and event identifier, without revealing the signer's identity [59].

Definition 3. An LRS scheme consists of five algorithms:

- $pp \leftarrow \text{LRS.Gen}(1^\lambda)$: Given the security parameter λ , it outputs the public parameters pp .
- $(sk, PK) \leftarrow \text{LRS.KeyGen}(pp)$: On input pp , it returns a pair of private and public keys (sk, PK) .

- $\sigma \leftarrow \text{LRS.Sign}(e_{ID}, sk_{I_r}, m, R)$: Given event identifier e_{ID} , ring member's secret key sk_{I_r} , message m , and public key list R , it outputs a signature σ .
- $\{0, 1\} \leftarrow \text{LRS.Verify}(e_{ID}, \sigma, m, R)$: 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.
- $\{0, 1\} \leftarrow \text{LRS.Link}(e_{ID}, \sigma, \sigma', m, m', R, R')$: Given e_{ID} , signatures (σ, σ') on messages (m, m') , and public key lists (R, R') , it returns 1 if linked, 0 otherwise.

We adopt a PQ secure event-oriented LRS scheme [59] built from a hash function and Signature of Knowledge (SoK) primitives (e.g., STARK-based SoK (ethSTARK) [53]) 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.

2.2.1. PQ-variant of The Onion Routing (PQ-Tor)

With the advent of NIST-standardized PQC, the PQ variant of the Tor network (PQ-Tor) retains the core architecture of conventional Tor while replacing vulnerable cryptographic components. Specifically, *AES128* is upgraded to *AES256* to mitigate Grover's algorithm [10]; RSA signatures used in consensus are replaced with ML-DSA [21]; RSA-based KEMs for circuit creation are substituted with ML-KEM [50]; 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 PQ-Tor proceeds as follows: (i) *PQ-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) *PQ-Tor.Circuit.Creation*: The client sends *CREATE* and *EXTEND* commands to establish the circuit, generating three AES keys and distributing them using ML-KEM encapsulation. (iii) *PQ-Tor.Send*($N_r; m$): transmits message m to receiver N_r via layered symmetric encryption, while *PQ-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.

3. Threat and Security Models

This section presents the adversarial and threat models of QPADL, specifying the attacker's capabilities, covered attack vectors, security assumptions, and system scope.

3.1. Threat Model

We consider a quantum-capable probabilistic polynomial-time (QPT) adversary in the QPADL system model. The adversary has full control over the wireless channel between clients and infrastructure entities, including eavesdropping, message injection, modification, replay, delay, and blocking. Malicious clients may also deviate arbitrarily from

the protocol by issuing adaptive or strategically crafted queries, reusing protocol artifacts, or submitting manipulated location-related information to obtain unauthorized spectrum access or service. We assume the spectrum regulator (e.g., FCC) is trusted as the root authority for system initialization and policy definition. Access points are certified infrastructure entities with valid long-term credentials and synchronized operation for generating proof artifacts.

Clients are assumed to hold a registration-time secret sk_c established during SAS enrollment, from which window-scoped pseudonyms are derived for per-client abuse control. The AP treats $n_{ym_{TW}}$ as opaque and does not verify its derivation from sk_c , so the scope of query-side rate limiting depends on one further capability of the malicious-client class: (C1) *Radio-layer identity churn*. A malicious client that is physically within AP coverage and able to change its radio-layer identity within a single time window can present distinct pseudonyms on distinct links, and thereby obtain more than one proof artifact in that window. Query-side rate limiting is per client against any adversary lacking (C1). Against an adversary with (C1), it degrades to per pseudonym at the PSDs and per link at the AP. The service-side counter-DoS guarantee is unaffected in either case, since it rests on puzzle cost rather than on artifact uniqueness (Definition 9). Section 5.3 gives a verifiable-derivation variant that removes (C1) cryptographically, at the cost of maintaining a client-side ring. The enrollment process itself lies outside our scope. Unless otherwise stated, PSDs and SAS servers are modeled as honest-but-curious: they correctly execute the protocol but may attempt to infer sensitive client information, such as identity, precise location (which, in the default flow, is never disclosed to the server and is opened only under defined accountability conditions; Section 5), or cross-session linkability, from protocol transcripts, metadata, or repeated interactions.

For the anonymous transport layer, we adopt the standard Tor adversary model: the adversary may operate or compromise individual relays and observe traffic on links it controls, but does not simultaneously control or observe both endpoints of a client's circuit, and does not collude with the PSD receiving that circuit's traffic. Consistent with this model, a global passive adversary observing the entire network, as well as end-to-end traffic-confirmation attacks based on timing, volume, or flow correlation, are outside our scope. Under this model, we consider the following representative attack classes:

- *Client Privacy and Anonymity Attacks*: Adversaries, including PSDs, SAS servers, and external observers, may attempt to infer a client's identity, location, or device-related attributes, or to link multiple sessions, by analyzing exchanged messages, metadata, or repeated protocol interactions. These attacks may involve passive observation, active probing, and correlation across the spectrum-query and service-access phases.
- *Location Spoofing Attacks*: Malicious clients may submit falsified coordinates or manipulated proximity-related information to gain unauthorized access to spectrum channels or SAS services outside their legitimate region.

Relevant attacks include fake location claims, replay of prior proof artifacts, relay-assisted manipulation, distance fraud, mafia fraud, distance hijacking, and timestamp manipulation [44, 43]. Replay and transfer attacks are countered cryptographically by the event-scoped proof artifact, and relay-based hijacking by RTT-weighted proximity verification, while attacks requiring light-speed distance bounding are outside our scope.

- **Denial-of-Service Attacks:** Malicious clients or external adversaries may attempt to degrade the availability of PSDs or SAS servers by flooding them with bogus, replayed, or computationally expensive requests during the spectrum-query or service-request phases. Such attacks aim to exhaust computational or communication resources, delay responses, or deny timely spectrum access to legitimate users.
- **Infrastructure Compromise:** A malicious or compromised AP may issue proof artifacts for clients not in its vicinity, and a leaked sk_{AP} permits forging artifacts within the affected ring. A malicious relay may drop, delay, or observe traffic on links it controls, and PSDs may collude to infer the queried index. Since $n_{ym_{TW}}$ is disclosed only to the AP, a malicious AP may additionally issue artifacts under arbitrary event scopes, uncapping the clients it serves; the impact remains confined to that AP's own region and is detectable through the resulting anomalous admission volume under a single ring.

QPADL covers colluding PSDs up to the privacy threshold t of the instantiated PIR, individually malicious relays, and malicious clients attempting spoofing, replay, or flooding. APs are regulator-certified; a misbehaving AP is nonetheless bounded in impact, as client coordinates stay hidden behind the commitment and the region-scoped beacon confines any improperly issued artifact to that AP's own region, while a compromised sk_{AP} remains detectable through event-scoped linkability and is recoverable by regulator-side revocation and ring re-formation. Collusion beyond t PSDs, colluding entry-exit relay pairs, relay-PSD collusion, global passive observers, and traffic-confirmation attacks are excluded.

3.2. 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 $|S| \leq t$, there exists a simulator $\mathcal{S}$ such that for all query indices $\{i_0, i_1\} \in DB$, $|\Pr[\mathcal{A}(\text{View}_S^{PIR}(i_0)) = 1] - \Pr[\mathcal{A}(\text{View}_S^{PIR}(i_1)) = 1]| \leq \text{negl}(\lambda)$, where $\text{View}_S^{PIR}(i)$ denotes the joint view of the corrupted servers in S when the honest client queries index i .

Definition 5 (v -Byzantine-Robust PIR). A PIR is v -Byzantine-robust if, for any set $B \subset [\ell]$ of at most v Byzantine servers and any index $i \in DB$, $\Pr[\text{BlockReconst}_{PIR}(\rho_{[\ell]}(i, B)) = DB[i]] = 1$, 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 $\theta \in DB$ and any subset $S \subseteq [\ell]$ with $|S| \geq k$, $\Pr[\text{BlockReconst}_{PIR}(\rho_S(\theta)) = DB[\theta]] = 1$, where $\rho_S(\theta)$ are the answers from servers in S .

Definition 7 (PQ-Tor Anonymity). Let $\text{Exp}_A^{\text{Anon}}(\lambda, b)$ be the experiment in which $\mathcal{A}$ selects two honest senders (u_0, u_1) and a receiver, may operate a subset of relays and observe links it controls, subject to not holding both endpoints of the challenge circuit; the challenger routes a message from u_b over PQ-Tor, and $\mathcal{A}$ outputs a guess b' . PQ-Tor provides anonymity if for every QPT $\mathcal{A}$, $\text{Adv}_A^{\text{Anon}}(\lambda) := \left| \Pr[b' = b] - \frac{1}{2} \right| \leq \text{negl}(\lambda)$, assuming all layers use PQ-secure KEMs and symmetric ciphers. This captures cryptographic sender-receiver unlinkability at the transport layer and does not model traffic-analytic deanonymization, which lies outside the adopted Tor adversary model.

Definition 8 (Proof-of-Location Soundness and Freshness). Let $\text{Exp}_{Adv}^{\text{PoL}}(\lambda)$ be the experiment in which Adv adaptively obtains proof artifacts from honest APs for locations and windows of its choice, and outputs $(C_{TW}^*, \sigma_{AP}^*, e_{ID}^*, TW^*)$. $\mathcal{A}$ wins if PoL.Verify accepts and either: (S) Soundness: the opening of C_{TW}^* places Adv at a location l^* outside the coverage region of every AP that issued an artifact to $\mathcal{A}$ in TW^* ; or (F) Freshness: $(\sigma_{AP}^*, e_{ID}^*)$ was issued in a window $TW' \neq TW^*$, or was already admitted under e_{ID}^* within TW^* .

The scheme provides proximity-backed location assurance if for every QPT Adv , $\Pr[\text{Exp}_{Adv}^{\text{PoL}}(\lambda) = 1] \leq \text{negl}(\lambda)$, under the proximity model of Section 4.1. Correctness requires that an honest client at (l_x, l_y) within AP coverage obtains an accepting artifact with probability $\geq 1 - \text{negl}(\lambda)$. Condition (S) is relative to the proximity model: adversaries achieving sub-region distance manipulation under light-speed-bounded timing lie outside this experiment.

Definition 9 (Counter-DoS). Let $\text{Exp}_A^{\text{DoS}}(\lambda)$ be the experiment in which $\mathcal{A}$ issues at most $q(\lambda)$ service or query requests, possibly replaying artifacts or submitting malformed puzzle solutions, while honest clients issue requests concurrently. $\mathcal{A}$ wins if some honest request is delayed beyond a fixed bound Δ while $\mathcal{A}$ expends computational cost below $q(\lambda) \cdot \tau$, where τ is the per-request puzzle cost. Here $\text{Delay}(\mathcal{A})$ denotes the maximum queuing-plus-service delay incurred by any honest request during the experiment, and $\text{Cost}(\mathcal{A})$ denotes the total computational effort $\mathcal{A}$ expends in producing its requests. A system is DoS-resilient if for every QPT $\mathcal{A}$, $\Pr[\text{Delay}(\mathcal{A}) > \Delta \wedge \text{Cost}(\mathcal{A}) < q(\lambda) \cdot \tau] \leq \text{negl}(\lambda)$, with τ enforced jointly by client puzzles and event-scoped rate limiting.

3.3. Scope of Our Solution

The QPADL framework is designed to protect client privacy during the spectrum-query and access phases. In particular, it aims to preserve client anonymity and location privacy while resisting location-spoofing attempts and mitigating DoS attacks in a PQ setting. These protections apply during spectrum querying, cryptographic puzzle retrieval, and subsequent service requests, with location

privacy preserved against both PSDs and the SAS server by default. Regulatory accountability conditions, dispute resolution, or flagged suspicious behavior during service requests may require location disclosure and are outside the privacy guarantees of this work. Our scope is limited to the query and access stages of the SAS workflow. We do not address privacy or security issues arising during user registration, long-term identity management, or actual spectrum utilization after authorization. PU privacy is also outside the scope of this work. In addition, QPADL does not address location leakage during ongoing spectrum usage, user mobility or handover scenarios [28], nor does it claim protection against side-channel leakage, timing analysis, end-to-end traffic-confirmation or flow-correlation attacks against the anonymous transport, or physical-layer localization techniques like triangulation [7]. Finally, our proof mechanism provides *proximity-backed location assurance* within the assumed system model and is not intended to constitute a complete defense against all real-world RF-layer localization or relay adversaries. Relatedly, verifiable derivation of the window-scoped pseudonym from the registration secret is not enforced in the default construction, whose guarantee is stated in Section 3.1; an optional variant that enforces it cryptographically is given in Section 5.3.

4. The Proposed Framework: QPADL

We outline the initial setup of QPADL framework, followed by its main operations, detailed algorithmic descriptions, and various instantiations and optimization strategies. We have introduced the description of the main notations of QPADL in Table 2.

4.1. QPADL Framework Initial Setup

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

Database Structure and Setup: The DB structure in QPADL follows FCC spectrum-sharing requirements, with each PSD maintaining synchronized entries indexed by a subroutine $\text{DB.Index}(\cdot)$ that maps tuples $((l_x, l_y), \text{ch}, \text{TV})$, consisting of grid-based location coordinates, spectrum channel, and validity window, to specific database blocks. The database is modeled as an $r_{\text{DB}} \times s_{\text{DB}}$ matrix, where each row represents a b -bit block partitioned into s_{DB} words over $GF(2)$ (or, more generally, $GF(2^w)$ depending on the PIR instantiation). In addition to location, indexing may incorporate device characteristics such as transmission power, antenna height, and device category, as well as access preferences including bandwidth and requested duration, thereby supporting accurate and policy-compliant spectrum decisions. This design is consistent with FCC requirements for CBSD geolocation accuracy, which impose bounded horizontal and, for certain device classes, vertical positioning constraints to preserve reliable spectrum assignment and cross-database consistency. The indexing mechanism may be instantiated using FCC-compatible spatial encodings such as geohash, H3, or S2 geometry to support efficient spatial lookup [1, 37]. After resolving the appropriate index,

Table 2

Summary of notations used in QPADL.

Notation	Description
λ, κ	Cryptographic security parameter and client-puzzle difficulty, respectively.
n, ℓ, t, v, k	Number of PSDs queried, number of responsive PSDs, PIR privacy threshold, tolerated Byzantine PSDs, and minimum responses required for block reconstruction.
$r_{\text{DB}}, s_{\text{DB}}, b, w$	Number of database rows, words per row, block size in bits, and word size over $GF(2^w)$.
$(l_x, l_y), \text{ch}, \text{TV}, \theta$	Client location, requested channel, validity interval, and the corresponding private database index.
$n_{\text{AP}}, \mathcal{R}_{\text{AP}}$	Number of APs in a region and their public-key ring used for the PQ-LRS.
$\text{TW}, \beta_{\text{TW}}$	Duration of the PoL and rate-limiting window and its region-scoped beacon.
$r_{\text{AP}}, \epsilon_{\text{est}}, \Delta_{\text{th}}$	AP coverage radius, ranging-error bound, and proximity-acceptance threshold, where $\Delta_{\text{th}} \geq r_{\text{AP}} + \epsilon_{\text{est}}$.
$v_{\text{max}}, \delta_{\text{FCC}}$	Maximum client speed and regulatory position budget ($\delta_{\text{FCC}} = 50 \text{ m}$), constraining the window via $v_{\text{max}} \cdot \text{TW} \leq \delta_{\text{FCC}}$.
H, H', sk_c, r_c	Hash functions, client registration secret, and fresh λ -bit nonce used to construct the commitment and window-scoped pseudonym.
$n_l, n_\Lambda, p, \alpha$	Number of HCT leaves and the lattice dimension, prime modulus, and approximation factor of the LBP puzzle.
$\mathcal{L}_{\text{TW}}$	Per-window admission log at each PSD, storing accepted $(e_{\text{ID}}, \sigma_{\text{AP}})$ pairs; flushed at window rollover.
$\mathcal{L}_{\text{AP}}$	Per-window link log at each AP, storing link handles (lnk, TW) already served in the current window; flushed at window rollover.

the DB.Record function stores the associated puzzle and its signature in the corresponding database entry.

Access Points Setup: APs in a region collectively form a ring $\mathcal{R}_{\text{AP}}$ containing the public keys of all participants, where each AP holds a key pair $(sk_{\text{AP}}, PK_{\text{AP}})$ generated by the FCC using $\text{LRS.KeyGen}(pp)$ algorithm (Definition 3). They periodically broadcast time-sensitive beacons (β_{TW}) for device discovery within a time window (TW) . The beacon β_{TW} is a region-scoped, time-dependent token without AP-specific identifiers. To derive a confidence-bounded proximity decision, an AP evaluates signal strength and round-trip time using environmental parameters [48, 40], computing the distance as $\Delta \leftarrow \text{ProxVerif}(\text{RSS}, \text{RTT}, \text{env}_{\text{params}})$.

Admission Control at the AP: Because ProxVerif is evaluated over a live radio link on which the AP already measures RSS and RTT, the AP issues at most one proof artifact per link per time window, keyed by the same link handle used for the proximity measurement (Algorithm 2, Steps 9-10). This introduces no information the AP does not already observe, and it bounds artifact issuance to adversaries with (C1) (Section 3). Cryptographic enforcement

is applied downstream, where the client is anonymous: at the PSDs via the event-scoped admission log, and at the service layer via the client puzzle.

Proximity Model and Assumptions: We model the `ProxVerif` as a proximity test mapping RSS and RTT measurements to a fused distance estimate Δ_c with bounded error ϵ_{est} , accepting when $\Delta_c \leq \Delta_{th}$. We assume: (i) a log-distance path-loss channel with log-normal shadowing, under which RSS yields a distance estimate with error captured by ϵ_{est} ; (ii) RTT measured with loosely synchronized AP clocks, sufficient for coarse ranging at region granularity but not for light-speed distance bounding; and (iii) beacons broadcast per time window, so freshness is enforced at TW granularity rather than by round-trip nonce latency. Under these assumptions, `ProxVerif` provides proximity-backed evidence that a requester lies within the AP's coverage region, which the LRS-based artifact then binds cryptographically to the query. We do not claim resistance to sub-region distance manipulation by adversaries with light-speed-bounded timing precision requiring dedicated distance-bounding hardware, which lies outside our scope.

Time-Window and Threshold Selection: Two parameters must be fixed, and they are governed by distinct constraints. The acceptance threshold Δ_{th} governs the proximity test itself: since the AP ranges the device over the live link at request time, the estimate reflects the device's current position, and the threshold need only absorb the coverage radius and the ranging error, i.e. $\Delta_{th} \geq r_{AP} + \epsilon_{est}$. With the RSS/RTT ranging error of our channel model ($\epsilon_{est} \approx 0.1 r_{AP}$ over $r_{AP} \in [30, 100]$ m), this yields $\Delta_{th} \approx 1.1 r_{AP}$, the operating point calibrated empirically in Section 7.2.4.

The window duration TW instead governs position staleness: a client commits to (l_x, l_y) upon beacon reception and may displace by up to $v_{max} \cdot TW$ before the artifact is consumed, so TW must keep this displacement within the regulatory position budget, $v_{max} \cdot TW \leq \delta_{FCC}$. With $\delta_{FCC} = 50$ m, the horizontal position accuracy that the FCC's Part 96 rules (§ 96.39(a)(1) which govern the CBRS band), require CBSDs to maintain [26], the rule yields $TW \leq \delta_{FCC}/v_{max} \approx 1.67$ s at highway speed ($v_{max} \approx 30$ m/s). We therefore adopt $TW = 1.5$ s for vehicular deployments, bounding worst-case displacement at 45 m and retaining margin within the position budget. For pedestrian and fixed deployments ($v_{max} \leq 1.5$ m/s), a relaxed window of 10-30 s satisfies the same rule, keeping displacement within 45 m. The artifact is then also refreshed roughly forty times faster than the 60-second deadline within which § 96.39(a)(3) requires location changes exceeding ± 50 m to be reported, so an artifact never outlives the regulatory obligation it attests to. A substantially longer vehicular window, e.g. $TW = 10$ s, would admit 300 m of position staleness, decoupling the attested grid cell from the device's actual location well beyond the accuracy the regulation assumes.

Granularity of the Proximity Binding: The AP's proximity check binds a request to the region scoped by the beacon β_{TW} embedded in the client's commitment, rather than to the exact coordinates, which remain hidden from the AP. This granularity is a deliberate design choice aligned with the operational reality of the SAS architecture. Spectrum

authorizations in SAS/CBRS are intrinsically location- and region-bound: the SAS assigns channels, power levels, and validity intervals as a function of a device's position relative to incumbents, protection points, and dynamic exclusion zones, and these decisions are themselves resolved over grid- or region-indexed database entries. Consequently, a grant obtained for one region carries no advantage at a different physical location, meaning its channel and power parameters are computed for the claimed region and are inapplicable, and typically physically unusable, elsewhere [1].

Moreover, actual transmission is governed by FCC-mandated physical enforcement independent of the query: Environmental Sensing Capability (ESC) detection, dynamic protection-area move lists, and aggregate-interference limits regulate emissions at the device's true position, so a client cannot convert a mismatched location claim into usable spectrum [13]. The residual and meaningful threat is therefore a client falsely claiming presence at a region it does not occupy in order to obtain spectrum there; QPADL addresses precisely this via the AP's `ProxVerif` check, which ties the request to the client's actual vicinity, together with the region-scoped beacon that binds the commitment to that vicinity. Coordinate-level cryptographic binding would therefore add no protection beyond what the enforcement model already provides, while requiring the AP to learn the precise position being attested.

This granularity is further consistent with the regulations themselves, which require CBSDs to determine their coordinates only to within ± 50 m horizontally and to report location changes exceeding this threshold within 60 seconds. Our per-time-window proof artifact mirrors this model for nomadic and vehicle-mounted deployments: upon relocating, such a device acquires a fresh proof from a nearby AP for the current time window and issues a new query for its updated grid cell, re-establishing proof freshness and spectrum authorization together, while continuous in-motion handover remains outside our scope (Section 3.3). QPADL thus provides region-scoped, proximity-backed location assurance as the appropriate binding granularity for SAS, with any coordinate-level verification deferred to the optional, policy-triggered opening described in Section 5.

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 2.2.1.

4.2. QPADL Framework Main Operations

QPADL Overview. QPADL realizes private, verified, and abuse-resistant spectrum access through a single four-phase workflow (Figure 1). During setup, each PSD binds a signed puzzle to every spectrum entry. A client then obtains a proximity-backed proof of location from a nearby AP and issues a private query over PQ-Tor, so that location privacy and communication anonymity are preserved simultaneously. Each PSD verifies the proof and, using its event-scoped linkability, enforces one-use-per-window rate limiting before returning its PIR response, reusing the same

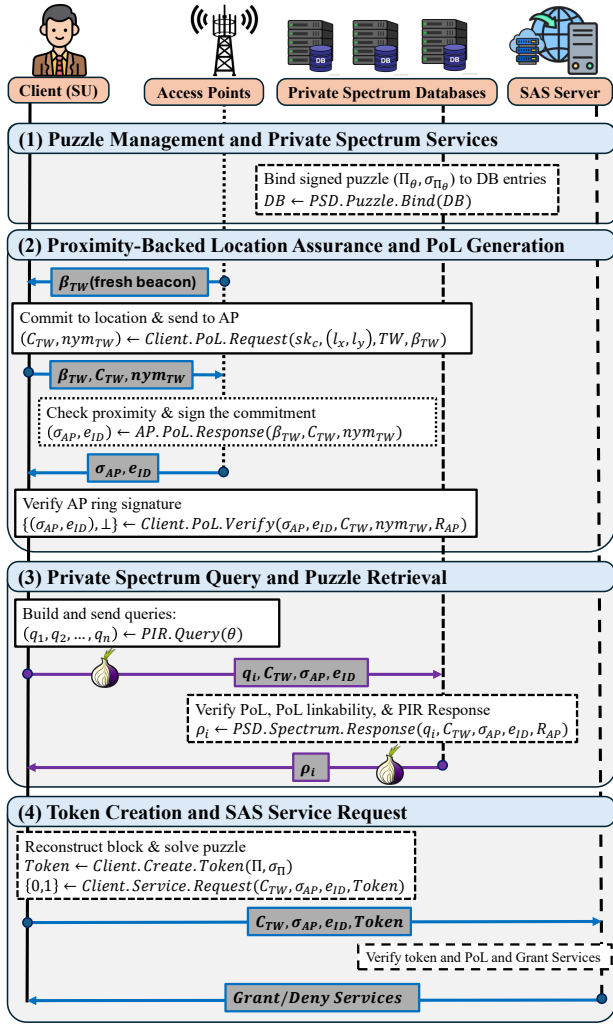


Figure 1: QPADL end-to-end message flow across the four operational phases.

artifact for both admission and DoS defense. Finally, the client reconstructs the block, solves the retrieved puzzle into a service token, and presents it with the proof for access. The synergy is deliberate: a location commitment and an event-scoped, linkable proof are threaded through all phases, so each security property reinforces the others rather than being bolted on; most notably, linkability provides event-scoped abuse control that anonymity would otherwise preclude. Algorithms 1-2 detail QPADL's core operations.

1) Puzzle Management and Private Spectrum Services: Each PSD initializes and maintains a synchronized spectrum DB containing spectrum-availability information together with pre-generated cryptographic puzzles and their signatures. For each admissible DB index, derived from location coordinates, channel, validity window, and relevant device attributes (e.g., power level, height, or category), the PSD generates a puzzle at the prescribed difficulty level and signs it using $ML\text{-}DSA.Sign$. The resulting records are stored under $DB.Index((l_x, l_y), ch, TV)$ and refreshed

Algorithm 1 QPADL Framework

1 Setup [PSD]: Bind Signed Puzzles to DB Entries

```

DB ← PSD.Puzzle.Bind(DB)
1: for all  $(l_x, l_y)$  do
2:   for all ch do
3:     for all TV do
4:       Given  $\theta \leftarrow DB.Index((l_x, l_y), ch, TV)$ 
5:        $\Pi_\theta \leftarrow Puzzle.Gen(1^\lambda, \kappa)$ 
6:        $\sigma_{\Pi_\theta} \leftarrow ML\text{-}DSA.Sign(sk_{PSD}, \Pi_\theta)$ 
7:        $DB.Record(\Pi_\theta, \sigma_{\Pi_\theta})$ 
    
```

2 Query [Client]: Obtain PoL, Build & Send PIR Queries

```

 $\{q_i\}_{i=1}^n \leftarrow Client.Spectrum.Query((l_x, l_y), ch, TV, TW, \beta_{TW})$ 
8:  $(C_{TW}, \sigma_{AP}, e_{ID}) \leftarrow Client.PoL(sk_c, (l_x, l_y), TW, \beta_{TW})$ 
9:  $\theta \leftarrow DB.Index((l_x, l_y), ch, TV)$ 
10:  $(q_1, q_2, \dots, q_n) \leftarrow PIR.Query(\theta)$ 
11: for  $i = 1, \dots, n$  do
12:    $PQ\text{-}Tor.Send(PSD_i; q_i, C_{TW}, \sigma_{AP}, e_{ID})$ 
    
```

3 Response [PSD]: Verify PoL & Linkability, PIR Response

```

 $\rho_i \leftarrow PSD.Spectrum.Response(q_i, C_{TW}, \sigma_{AP}, e_{ID}, R_{AP})$ 
13:  $PSD.PQ\text{-}Tor.Receive(SU; q_i, C_{TW}, \sigma_{AP}, e_{ID})$ 
14: if  $1 = PoL.Verify(e_{ID}, \sigma_{AP}, C_{TW}, R_{AP})$  then
15:   if  $e_{ID} \in \mathcal{L}_{TW}$  then, return  $\perp$ 
16:   else if  $1 = LRS.Link(e_{ID}, \sigma_{AP}, \sigma'_{AP}, C_{TW}, C'_{TW}, R_{AP}, R_{AP})$  for
       some  $\sigma'_{AP} \in \mathcal{L}_{TW}$  then, return  $\perp$ 
17:   else  $\mathcal{L}_{TW} \leftarrow \mathcal{L}_{TW} \cup \{(e_{ID}, \sigma_{AP})\}$ 
18:    $\rho_i \leftarrow PIR.Query.Response(q_i, DB)$ 
19:  $PSD.PQ\text{-}Tor.Send(SU; \rho_i)$ 
    
```

4 Token [Client]: Reconstruct Block, Solve Puzzle

```

Token ← Client.Create.Token( $\Pi, \sigma_\Pi$ )
20: for  $i = 1, \dots, k$ , where  $k \in [t, n]$  do
21:    $Client.PQ\text{-}Tor.Receive(PSD_i; \rho_i)$ 
22:    $DB_\theta \leftarrow Client.PIR.BlockReconst(\theta, (\rho_1, \rho_2, \dots, \rho_k))$ 
23:   if  $1 \leftarrow ML\text{-}DSA.Verify(PK_{PSD}, \Pi, \sigma_\Pi)$  then
24:      $\Psi \leftarrow PoW(\Pi, \kappa)$ 
25:     return Token ←  $(\Pi, \sigma_\Pi, \Psi)$ 
    
```

5 Access [CRN Server]: Verify Token & PoL, Grant Service

```

 $\{0,1\} \leftarrow Client.Service.Request(C_{TW}, \sigma_{AP}, e_{ID}, Token)$ 
26: if  $1 = ML\text{-}DSA.Verify(PK_{PSD}, \Pi, \sigma_\Pi)$ , then
27:   if  $1 = PoW.Verify(\Pi, \Psi)$  then, Record the Token
28:   if  $1 \leftarrow PoL.Verify(e_{ID}, \sigma_{AP}, C_{TW}, R_{AP})$ , then
29:     return 1, and grant access.
    
```

according to their validity interval (e.g., hourly). The number of puzzles generated depends on device characteristics, server count, and their maximum capacity [1]. This setup enables clients to retrieve both spectrum information and a signed anti-DoS puzzle through the same privacy-preserving query process.

2) Proximity-Backed Location Assurance and PoL Generation: Before issuing a spectrum query, the client obtains a fresh proof artifact for the current time window TW from a nearby AP, as shown in Algorithm 2. Upon receiving the latest beacon β_{TW} , the client computes a commitment C_{TW} to its location, the current window, and a random nonce, i.e., $C_{TW} = H((l_x, l_y) || \beta_{TW} || TW || r_c)$. The commitment is hiding: only C_{TW} is disclosed during proximity verification and query admission, while the coordinates (l_x, l_y) and

nonce r_c remain concealed from the AP and PSDs, and are opened only at service access under the conditions of Section 5. The client then sends $(\beta_{TW}, C_{TW}, \text{nym}_{TW})$ to the selected AP. The AP first checks beacon freshness and then applies a proximity-verification procedure based on local signal measurements (e.g., RSS and RTT) to determine whether the requester is plausibly within the AP's vicinity. If this check succeeds, the AP derives an event identifier e_{ID} from the AP ring, the current beacon, and the active time window, and produces a ring signature over the client's commitment. The client verifies this signature and accepts (σ_{AP}, e_{ID}) as a valid proof artifact for subsequent query admission.

Algorithm 2 $(C_{TW}, \sigma_{AP}, e_{ID}) \leftarrow \text{PoL}(sk_c, (l_x, l_y), TW, \beta_{TW})$

1 PoL Request [Client]: Commit to Location, Send to AP
 $(C_{TW}, \text{nym}_{TW}) \leftarrow \text{Client.PoL.Request}(sk_c, (l_x, l_y), TW, \beta_{TW})$

- 1: Given the latest beacon β_{TW} :
- 2: $r_c \xleftarrow{\$} \{0, 1\}^\lambda$, $\text{nym}_{TW} \leftarrow H'(sk_c, TW)$
- 3: $C_{TW} \leftarrow H((l_x, l_y) || \beta_{TW} || TW || r_c)$
- 4: Send $(\beta_{TW}, C_{TW}, \text{nym}_{TW})$ to AP.

2 PoL Response [AP]: Check Proximity, Sign Commitment
 $(\sigma_{AP}, e_{ID}) \leftarrow \text{AP.PoL.Response}(\beta_{TW}, C_{TW}, \text{nym}_{TW})$

- 5: **if** β_{TW} is not the latest beacon: **then, return** $\perp$
- 6: **else**
- 7: $\Delta_c \leftarrow \text{ProxVerif}(\text{RSS}, \text{RTT}, \text{env}_{\text{params}})$
- 8: **if** $\Delta_c \leq \Delta_{th}$ **then**
- 9: **if** $(\text{lnk}, TW) \in \mathcal{L}_{AP}$ **then, return** $\perp$
- 10: $\mathcal{L}_{AP} \leftarrow \mathcal{L}_{AP} \cup \{(\text{lnk}, TW)\}$
- 11: $e_{ID} \leftarrow H'(R_{AP}, \beta_{TW}, TW, \text{nym}_{TW})$
- 12: $\sigma_{AP} \leftarrow \text{LRS.Sign}(e_{ID}, sk_{AP}, C_{TW}, R_{AP})$
- 13: Send (σ_{AP}, e_{ID}) to the Client.

3 PoL Verify [Client]: Verify AP Ring Signature
 $\{(\sigma_{AP}, e_{ID}), \perp\} \leftarrow \text{Client.PoL.Verify}(\sigma_{AP}, e_{ID}, C_{TW}, \text{nym}_{TW}, R)$

- 14: **if** $e_{ID} = H'(R_{AP}, \beta_{TW}, TW, \text{nym}_{TW})$ **then,**
- 15: **if** $1 = \text{LRS.Verify}(e_{ID}, \sigma_{AP}, C_{TW}, R_{AP})$ **then,**
- 16: **return** (σ_{AP}, e_{ID})

3) Private Spectrum Query and Puzzle Retrieval: After obtaining a valid proof artifact (Step 8), the client computes the target database index $\theta \leftarrow \text{DB.Index}((l_x, l_y), \text{ch}, \text{TV})$ and generates PIR queries for the participating PSDs (Steps 9-10). These queries are transmitted through PQ-TOR together with the committed proof tuple $(C_{TW}, \sigma_{AP}, e_{ID})$, thereby combining query privacy with communication unlinkability (Steps 11-12). Upon receiving a request, each PSD verifies the proof artifact and checks its event identifier against the admission log $\mathcal{L}_{TW}$ maintained for the current window. Since e_{ID} is derived from the region-scoped ring and beacon together with the client's window-scoped pseudonym, it is identical for every artifact obtained under that pseudonym within the window, irrespective of which AP in the region issued it, and distinct across pseudonyms. Reuse is therefore detected at the event scope itself, and the linkability tag additionally links artifacts issued by the same AP under that scope. If the proof is linked, the request is rejected according to the one-use-per-window policy;

otherwise, the proof is recorded and the PSD returns its PIR response (Steps 15-19). Once sufficiently many responses are collected, the client reconstructs the requested database block and extracts the corresponding spectrum information, puzzle, and puzzle signature. In this way, the same event-scoped proof supports both privacy-preserving query admission and lightweight rate limiting against repeated or abusive requests.

(4) Token Creation and SAS Service Request: After reconstructing the target DB entry, the client verifies the authenticity of the retrieved puzzle using the PSD's public key and then solves the puzzle to generate a service token (Steps 23-24). The token is subsequently submitted when requesting SAS-related services. Upon receiving the request, the server verifies the puzzle signature, checks the validity of the puzzle solution, and records the token to prevent replay or repeated computational abuse (Step 26). It then verifies the accompanying proof artifact to confirm that the request is bound to a fresh, proximity-backed commitment generated within the valid time window (Steps 27-28). If all checks succeed, the server grants service access. The optional, policy-triggered opening of the commitment under the accountability conditions is discussed in Section 5. This phase ensures that service requests are tied to both a valid spectrum-query outcome and a computationally bounded anti-DoS mechanism.

5. QPADL Framework Instantiations

This section presents concrete realizations of QPADL under different deployment and performance objectives. We first instantiate the query and service phases using specific PIR- and PoW-based constructions, then describe parallelization strategies for improving online response latency, and finally discuss practical optimizations that further improve deployability and efficiency.

5.1. Cryptographic Primitive Instantiation

We instantiate QPADL using concrete building blocks for the spectrum-query and service-request phases. In particular, we present three PIR-based query instantiations and two PoW-based service instantiations, each capturing a different trade-off among privacy assumptions, robustness, communication, and online efficiency, followed by a complete end-to-end realization shown in Figure 2.

5.1.1. 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 [19], 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 [19], with a GPU-parallelized version detailed in Section

5.2. This setup is best suited to settings where strict non-colluding synchronized PSDs is a reasonable deployment assumption and minimum client-side overhead is desired.

(ii) *QPADL-FTR*: This Fault Tolerant Robust (FTR) instantiation of QPADL adopts the IT-secure PIR scheme from [30], enabling v -Byzantine fault tolerance by allowing block reconstruction even if up to v servers return incorrect responses. Servers compute a query vector multiplication over the DB 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 $[t, n]$ PSDs. In cases of synchronization errors, transmission faults, or Byzantine behavior ($v < k$), the client employs Guruswami-Sudan list decoding for error correction, ensuring robustness. Further details of this PIR are provided in [30], with its parallelized implementation described in Section 5.2. Compared with QPADL-ENS, this instantiation trades additional reconstruction complexity for stronger resilience to faulty, inconsistent, or Byzantine PSD responses.

(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 [34], 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. [19], 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 PSDs maintain the same data, though chunk order may vary. Due to its favorable online efficiency and communication profile, QPADL-OOP serves as the representative query instantiation in our end-to-end workflow of Figure 2. Its use reflects a deployment-oriented design choice for large-scale SAS settings, while explicitly shifting from information-theoretic privacy guarantees to computational security.

5.1.2. PoW Instantiations

The service-request phase of QPADL can be instantiated with different PQ-secure PoW mechanisms depending on the desired trade-off between client-side cost, verification efficiency, and communication overhead.

(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 [5], HCT is designed to resist quantum and parallel brute-force attacks [18, 4], 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. For this reason, QPADL-HCT is the practical default in our full instantiation, especially when user devices have heterogeneous computational and energy budgets. The detailed HCT algorithm is presented below:

- $\Pi \leftarrow \text{HCT.Puzzle.Gen}(1^\lambda, \kappa)$: Randomly selects $n_s \xleftarrow{\$} \{0, 1\}^\lambda$ and sets the number of leaves n_l based on the difficulty level κ . The puzzle is $\Pi = (h, n_s, \kappa, n_l)$, where h is the hash function and n_s is the selected randomness.
- $\Psi \leftarrow \text{HCT.PoW}(\Pi)$: 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 nonce n_1 serves as the PoW token, with nonces $(n_{2^{n_l}-1}, \dots, n_1)$.
- $\{0, 1\} \leftarrow \text{HCT.PoW.Verify}(\Pi, \Psi)$: 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 [9], built on the hardness of the Shortest Vector Problem (SVP) [56]. This alternative offers security rooted in lattice hardness, but at a noticeably higher computational and communication cost than hash-based constructions, making it more suitable for settings where such overhead is acceptable. It comprises three core subroutines, described below:

- $\Pi \leftarrow \text{LB.Puzzle.Gen}(1^\lambda, 1^{n_\Lambda}, \kappa)$: Given security parameter λ , lattice dimension n_Λ , and difficulty level κ , the algorithm samples from uniform distribution $x_2, \dots, x_n \leftarrow \mathcal{U}(0 \cup [p-1])$, 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 $[p \ x_2 \ \dots \ x_n]$, ones on the subdiagonal, and zeros elsewhere. It outputs the puzzle $\Pi = (\alpha, n_\Lambda, B, p)$.
- $\Psi \leftarrow \text{LB.PoW}(\Pi, \kappa)$: Solves the puzzle Π by finding $v \in \Lambda(B)$ such that $\|v\| \leq \alpha \cdot p^{1/n_\Lambda}$, and returns $\Psi = (v, v)$ where $v = B \cdot v$.
- $\{0, 1\} \leftarrow \text{LB.PoW.Verify}(\Pi, \Psi)$: Verifies the solution by checking $\|v\| \leq \alpha \cdot p^{1/n_\Lambda}$, $v = B \cdot v$, and $v \in \mathbb{Z}^{n_\Lambda}$, returning 1 if all conditions hold.

Instantiation Rationale and Selection. The QPADL framework is designed to support multiple instantiation choices because different SAS deployments prioritize different goals, such as stronger information-theoretic query privacy, robustness to faulty or unresponsive DBs, or reduced online computation at scale. Accordingly, the PIR layer governs how spectrum information and signed puzzles are retrieved under different trust and performance assumptions, while the PoW layer governs how service requests are rate-limited under different client-side resource constraints. These instantiations are not independent additions; rather, they represent interchangeable design points within the same QPADL workflow, enabling the framework to be adapted to federated, robust, or large-scale deployment settings. Further details about the performance of each instantiation are provided in Section 7.

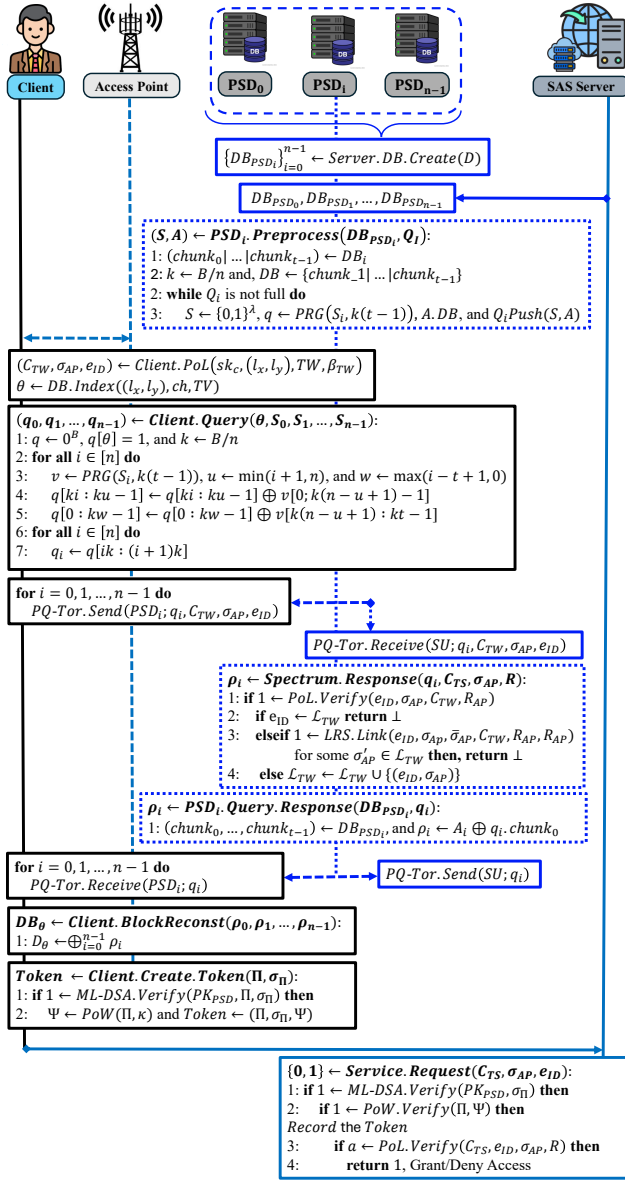


Figure 2: QPADL Full Instantiation

5.1.3. Full Instantiation of QPADL

Figure 2 illustrates a complete realization of QPADL using QPADL-OOP for privacy-preserving spectrum queries and QPADL-HCT for service-side request throttling. This instantiation is chosen as a representative deployment-oriented design point because it combines reduced on-line PSD computation with efficient client-verifiable PoW, while preserving the overall QPADL workflow and security goals. The DB contains frequency data $((l_x, l_y), ch, TV)$, with each entry bound to a signed puzzle $(\pi_\theta, \sigma_{\pi_\theta})$ via $\text{Puzzle.Bind}()$ (Algorithm 1, Steps 1-7). While all PSDs store the same content, their chunk orders may differ. Each block $D_\theta = ((l_x, l_y), ch, TV, \Pi_\theta, \sigma_{\Pi_\theta})$ is divided into B chunks, with $k = B/n$. For all $i \in [n]$, chunks are defined as $chunk_i = block_{ki} \dots block_{ki+k-1}$ and each PSD arranges its DB as $DB_{PSD_i} = chunk_i \dots chunk_{i+t-1 \bmod n}$.

The CIP-PIR scheme requires each 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 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 $(t-1)/n$ 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 event-scoped proof artifact in phase two. After receiving the latest beacon β_{TW} from a nearby AP, the client sends a location commitment as a PoL request. If the AP's proximity check succeeds, it derives an event identifier from the current beacon, AP ring, and active time window, signs the client's commitment, and returns (σ_{AP}, e_{ID}) (Algorithm 2, Steps 1-14). 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 PSD_i (Steps 1-7). Finally, the client privately transmits the proof artifact and queries $(q_i, C_{TW}, \sigma_{AP}, e_{ID})$ to each 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. Thus, the same event-scoped proof artifact serves two purposes at query time: it admits only fresh, proximity-backed requests and enforces one-use-per-window rate limiting through linkability under the same event identifier. Next, PSD_i retrieves its assigned flip chunk (e.g., $chunk_0$ in 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 the default access flow, the server grants service after verifying the puzzle signature, the PoW solution, and the proximity-backed PoL, without learning the client's plaintext coordinates: the commitment C_{TW} and the region-scoped beacon β_{TW} it embeds are sufficient to bind the request to a valid, in-region PoL for the current window, so location privacy is preserved against the SAS server. Precise coordinates are disclosed only under explicitly defined

accountability conditions such as regulatory or incumbent-protection audit, coordinate-dependent services (e.g., exact power-level assignment), or flagged abuse such as repeated linkability-tag collisions or token-replay attempts. When triggered, the client opens the commitment by revealing (l_x, l_y, r_c) over an authenticated channel (e.g., PQ-TLS), and the server accepts iff $C_{TW} = H((l_x, l_y) || \beta_{TW} || TW || r_c)$. This disclosure is optional and policy-driven; it is outside our privacy guarantees when invoked, and the secure channel establishment itself is beyond the scope of this work.

5.2. Instantiation via Parallelization

Our parallelization objective is to reduce online PSD response latency under concurrent user demand, since this component directly affects the practicality of privacy-preserving spectrum access in large-scale SAS deployments. Numerous GPU-accelerated works [34, 41, 42] have been proposed to improve the efficiency of PIR schemes. Gunther et al. [34] 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. [42] attempts to improve the efficiency of homomorphic encryption-based single-server PIR and therefore does not align with our design rationale to provide resiliency against rogue attacks and a single root of trust. We next summarize the relevant GPU execution model and then present parallelized realizations of the PIR response computation.

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* ($\approx$ 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* ($\approx$ 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 [39], 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.

Parallel Chor-PIR: The online PSD computational overhead consists of matrix multiplication over $GF(2)$ (i.e., $\rho \leftarrow q \cdot DB$), where $\rho \in \{0, 1\}^b$, $DB \in \{0, 1\}^{r \times b}$, and $q \in \{0, 1\}^r$. This operation consists of conditional aggregation (via bitwise XOR) of matrix rows DB based on the query's bit values $\{q_{r'}\}_{r' \in \{1, \dots, r\}}$. 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. [60] 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.

Our proposed parallel Chor-PIR is detailed in Algorithm 3. `Query.Response` processes multiple queries $q = \{q_i\}_{i=1}^{\bar{q}}$ in batches. First, it copies the queries q to global memory before invoking the CUDA kernel (Step 2). The DB is already residing in the global memory. The kernel is configured with $(\bar{q}, \lfloor \frac{b}{32} \rfloor)$ 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 $t_y = 0$ 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.

Algorithm 3 Multi-request Parallel Chor-PIR

```

 $\rho \leftarrow \text{PSD.Query.Response}(q)$ : Upon receiving multiple
requests  $q = \{q_i = \{q_{i,1}, q_{i,2}, \dots, q_{i,r}\}\}_{i=1}^{\bar{q}} \in_R GF(2)^r$ 
1: each  $\text{PSD}_i$  for  $i = 1, 2, \dots, \bar{\ell}$  do
2:   copy  $q$  from main memory to global memory
3:    $|grid| = (\bar{q}, \lfloor \frac{b}{32} \rfloor)$ ,  $|block| = (32, 8)$ 
4:   CUDA Kernel:
5:     for each block  $(b_x, b_y) \in \{(1, 1), \dots, |grid|\}$  do
6:       for each thread  $(t_x, t_y) \in \{(1, 1), \dots, |block|\}$  do
7:          $c \leftarrow 32 \cdot b_y + t_x$ ,  $a \leftarrow 0$ 
8:         for  $r' = t_y, \dots, r$  [step = 8] do
9:           if  $q_{b_x, r'} = 1$  then  $a \leftarrow a + DB_{r', c}$ 
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  $\rho = \{\rho_i = \{\rho_{i,1}, \dots, \rho_{i,b}\}\}_{i=1}^{\bar{q}}$ 

```

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 q , each processed over a DB. q and DB are of size $\bar{q} \cdot r$ and $r \cdot b$, respectively. For each incoming PIR query batch 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 DB 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).

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 ρ 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 [38] 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.

Algorithm 4 Multi-request Parallel Goldberg-PIR

```

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

5.3. Instantiation Optimizations

Offline-Online mode: The offline-online execution mode improves practicality across several phases of QPADL by shifting reusable computations away from the critical online path. (i) In the PoL phase, static ring configuration enables reusable preprocessing and reduces repeated online work. (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 PSD servers. (iii) In the service request phase, the HCT-based PoW can be converted to a non-interactive form via the Fiat-Shamir heuristic [14], enabling users to generate verifiable tokens independently without interactive communication.

Database Compression Techniques: Since DB size directly affects PSD computation, particularly in PIR, compression can substantially improve spectrum query performance in QPADL (see Section 7). QPADL employs a technique that sorts DB entries (e.g., FCC frequency data) and stores differences between successive items instead of full values [52]. 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 $r_{DB} \times b$ bits to about $O(r_{DB}(b - \log(r_{DB})))$ bits. The resulting efficiency gains in computation and storage are detailed in Section 7. This optimization reduces storage and response-side processing cost without changing the privacy guarantees of the underlying PIR construction.

Multiple Block Retrieval: To reduce repeated query overhead, a single DB block may store multiple puzzles of increasing difficulty, enabling the client to retrieve several service-use tokens within one PIR query. This optimization preserves index indistinguishability because the queried block index remains hidden by the underlying PIR, even when multiple puzzles are embedded in the same block. However, as discussed earlier, retrieving several puzzles at once may reduce future query frequency and thus expose limited coarse-grained temporal behavior at the service layer; it does not reveal which DB block was queried. Accordingly, this optimization improves communication efficiency while preserving query privacy at the PIR layer.

Verifiable Pseudonym Derivation: Where a deployment requires the AP to verify that nym_{TW} is bound to a registered secret, QPADL admits a drop-in upgrade using the same primitive it already employs. Registered SUs form a client-side ring R_c , and the client accompanies its commitment with an event-oriented LRS under event scope $e_c = H'(R_c, \beta_{TW}, TW)$, whose tag $T_c = H'(sk_c, e_c)$ replaces nym_{TW} . The SoK underlying the LRS already proves in zero knowledge that the tag is correctly computed from a secret whose public key is a leaf of the ring, so correct derivation is verifiable without an additional proof system, and PQ security is inherited from the same hash-based construction. The tag is client-specific, AP-independent, unforgeable, and unlinkable across windows, yielding cryptographic rather than physical enforcement at the AP. The cost is one additional client-side LRS signature, whose signing time grows only logarithmically in the ring size and remains on the same order as the existing client-side PoL cost.

We retain this construction as an optional variant rather than the default, since maintaining R_c at SAS scale introduces costs and trust assumptions the default flow avoids. Enrollment and revocation are continuous across a large SU population, so the ring digest changes frequently and every update must be redistributed to APs and clients before artifacts under the prior digest are rejected; the hash-based tag additionally offers no native revocation, so a compromised sk_c remains admissible until the ring is republished. Whoever maintains the ring becomes a trust anchor whose compromise silently admits fabricated tags, which is precisely the class of infrastructure compromise our threat model treats explicitly. Deployments that require

the stronger guarantee and can absorb this ring-management overhead may adopt the variant.

Overall, these instantiations show that QPADL is not tied to a single cryptographic realization; rather, it defines a common SAS access workflow adaptable to different privacy, robustness, and efficiency requirements. The chosen end-to-end instantiation represents one operating point, while the broader design space highlights how the framework can be tuned for different deployment environments.

6. Security Analysis

We present a sequence of security proofs aligned with the threat, security, and system models. Below, we state the main theorems and lemmas for QPADL; the full analysis and detailed proofs appear in Appendix A.1.

Theorem 1. *For any QPT adversary interacting with QPADL, the framework achieves, except with probability $\text{negl}(\lambda)$: (i) PQ-secure location private spectrum access, where the privacy of the queried spectrum index reduces to the security of the instantiated PIR layer, namely IT-secure for QPADL-ENS and QPADL-FTR, and computationally-secure for QPADL-OOP under the hardness of its underlying PRG and preprocessing model. (ii) PQ communication anonymity and unlinkability, where the confidentiality of the anonymous transport reduces to the IND-CPA security of the layered encryption and the PQ security of the KEM, relying on standard symmetric-key assumptions together with Module-LWE-based hardness. (iii) Authenticity and pseudonym-scoped event linkability of the proof artifact, where the validity and misuse-detection properties of the proximity-backed proof artifact reduce to the simulation-extractability and soundness of the underlying SoK together with the one-wayness, collision resistance, and second-preimage resistance of the employed hash functions, applied to the event-identifier derivation. (iv) Authentic and abuse-resistant service access, where puzzle authenticity reduces to the EUF-CMA security of ML-DSA under standard module-lattice assumptions (e.g., Module-SIS/Module-LWE), while service-side computational throttling and replay/rate-limit enforcement reduce to the hardness of the selected puzzle construction, the hash-based binding of tokens, commitments, and event identifiers, and the collision resistance of H and H' underlying event-identifier uniqueness. Rate-limit enforcement is per pseudonym, and per client against an adversary lacking (C1) (Section 3.1).*

Lemma 1. *QPADL-ENS and QPADL-FTR achieve IT-secure private spectrum retrieval, where QPADL-ENS guarantees $(\ell-1)$ -private query-index privacy and QPADL-FTR guarantees t -private query-index privacy together with v -Byzantine robustness and k -out-of- ℓ correctness under the Shamir-secret-sharing and decoding conditions.*

Lemma 2. *QPADL-OOP achieves computationally secure private spectrum retrieval against polynomial-time adversaries, where query-index privacy reduces to the pseudorandomness of the underlying PRG together with the preprocessing and chunk-permutation assumptions of the construction.*

Lemma 3. *QPADL-HCT provides hash-based computational throttling for service access, where the adversary's success probability in bypassing the required client work reduces to the second-preimage resistance of the underlying hash function and the sequential structure of the HCT.*

Lemma 4. *QPADL-LBP provides lattice-based computational throttling for service access, where the adversary's success probability in bypassing the required client work reduces to the hardness of the underlying α -Hermite Shortest Vector Problem instance family.*

7. 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.

7.1. Configuration and Experimental Setup

Hardware: We performed our benchmarks on various hardware platforms. *(i) CPU platform:* We evaluated the efficiency of the QPADL framework using a standard desktop equipped with an Intel Core i9-11900K processor operating at 3.50 GHz, 64 GiB of RAM, a 1 TB SSD, and Ubuntu 22.04.4 LTS. *(ii) GPU platform:* The desktop was also equipped with an NVIDIA GeForce GTX 3060 GPU featuring 3,584 CUDA cores, 12 GB of GDDR6 memory, and a memory bandwidth of 360 GB/s. *(iii) Low-end platforms:* We also evaluated QPADL's client-side operations on an ARM Cortex-A72-based Raspberry Pi 4 (Model B) with a 64-bit quad-core processor and 8 GB of LPDDR4 RAM.

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 repository³ for the ring signature and the hashcash-tree repository⁴ for the hash-based puzzle. Additionally, DBs were constructed using *SQLite*⁵. For CPU-bounded implementation, we consider AVX instructions and OpenMP [16] as in [34].

Evaluation Metrics and Rationale: We assess 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

¹<https://percy.sourceforge.net/>

²<https://openquantumsafe.org/>

³<https://github.com/yuxil16/>

Post-Quantum-Linkable-Ring-Signature?tab=

⁴<https://github.com/alviano/hashcash-tree>

⁵The DB is modeled as a matrix with adjustable row counts (e.g., 2^{12} , 2^{14} , 2^{16} , 2^{18}). For each grid cell (l_x, l_y) , we generated synthetic spectrum records and embedded signed HCT and LBP puzzles, storing them across PSDs synchronized per FCC requirements [33].

Table 3

Parameters of the ProxVerif evaluation.

Parameter	Value range	Step
Path-loss exponent	3.0	—
Shadowing std. dev.	4 dB	—
RTT ranging jitter	3 m	—
AP coverage radius r_{AP}	30-100 m	—
Proximity threshold Δ_{th}	$1.1 r_{AP}$	—
Attacker distance from AP	50-300 m	10 m
Relay (benign) UE distance	0-50 m	10 m
Malicious UE distance	60-160 m	20 m
RTT fusion weight w_{RTT}	0.1-0.9	0.1
Trials per configuration	100	—

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.

Parameter Selection. We recall QPADL's parameters in Table 2. All our parameters target NIST Level I (128-bit classical and its post-quantum equivalent). PQ-Tor uses AES-256 for layered encryption and Level I ML-KEM for per-hop encapsulation; puzzle signatures use ML-DSA-44 with a 2420-byte signature, and the LRS uses Rescue-Prime inside an ethSTARK-based SoK.

PIR servers are set to $\ell = 3$ non-colluding PSDs. QPADL-ENS operates over GF(2) and is information-theoretically $(\ell-1)$ -private, QPADL-FTR operates over GF(2⁸) with $(t, k) = (1, 3)$ and is information-theoretically t -private, and QPADL-OOP inherits 128-bit post-quantum security from its SHA-256-CTR PRG under Grover.

Database and Puzzles is modeled as an $r \times b$ matrix with $r \in \{2^{12}, 2^{14}, 2^{16}, 2^{18}\}$ rows and $b = 3$ KB per block, spanning 8 MB to 512 MB to cover realistic CBRS spectrum tables. Moreover, Puzzles are instantiated in two variants. QPADL-HCT uses a hash-cash tree with $n_l = 2$ leaves and $\kappa \in \{14, 18, 20, 23\}$. QPADL-LBP uses lattice dimension n_Λ with $\alpha = 1.05 \cdot \Gamma(n_\Lambda/2 + 1)^{1/n_\Lambda} / \sqrt{\pi}$, solve cost $2^{0.2925 n_\Lambda + o(n_\Lambda)}$ classically and $2^{0.2570 n_\Lambda + o(n_\Lambda)}$ against Grover-accelerated sieving, with n_Λ chosen so LBP hardness fall in the same choices of κ .

LRS ring sizes span $n_{AP} \in \{2^3, \dots, 2^{13}\}$, covering local AP coalitions through dense metropolitan deployments. *Proximity and timing* parameters are set as follows: the time window is 1.5 s for vehicular clients ($v_{max} \approx 30$ m/s), consistent with the rule $TW \leq \delta_{FCC}/v_{max}$ of Section 4.1, and 10-30 s for pedestrian or fixed clients ($v_{max} \leq 1.5$ m/s), with $\Delta_{th} = 1.1 r_{AP}$, consistent with the rule $\Delta_{th} \geq r_{AP} + \epsilon_{est}$ of Section 4.1 under the ranging error of our channel model, and $r_{AP} \in [30, 100]$ m spanning typical small-cell and WiFi coverage in dense CBRS deployments. Channel parameters follow Table 3.

Open-Source Artifact. We provide a complete implementation of QPADL, including CPU and GPU implementations

of the PIR schemes, as well as puzzle-generation tools. The source code is available at:

<https://github.com/kiarashsedghigh/GPU-Multiserver-PIR-PuzzleGen.git>

7.2. Experimental Results

The analytical and empirical cryptographic, computational, and communication overhead across each phase of QPADL is shown in Tables 4 and 5 and elaborated below:

7.2.1. Computational Costs

(i) *PoL Phase:* On the user side, this phase involves generating a location commitment and a window-scoped pseudonym via hashing and verifying the LRS; the additional pseudonym derivation requires a single hash evaluation ($< 1 \mu s$) and is negligible relative to the 33.34 ms client-side PoL cost, 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 SAs. Verification is lightweight, 0.4 ms for 2^3 users and 8 ms for 2^6 , and ProxVerif adds only 1-10 ms using RSS 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 5.

(iii) *Spectrum Query Phase:* This phase integrates three PIR schemes with 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. Admission-log maintenance adds one hash-table lookup and insertion per request, with storage linear in the number of admitted clients per region and window, and is negligible relative to the measured PIR response cost.

(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 20. In contrast, LBP employs lattice basis reduction and enumeration [2], with runtimes ranging from 133-881 ms. For larger κ ,

Table 4

Analytical computational costs and communication overhead of QPADL.

Phase	Entity	Computational Cost	Communication Cost
PoL QPADL	User	$O(n_{\text{AP}}) \cdot H' + O(\log \log(n_{\text{AP}})) \cdot H'$	$O(\text{polylog}(\log(n_{\text{AP}})))$
	AP	$O(n_{\text{AP}}) \cdot H' + O(\log(n_{\text{AP}}) \cdot \log \log(n_{\text{AP}})) \cdot H'$	
Spectrum Query			
Puzzle.Bind	User PSD	$16r \cdot O(n_\sigma \log n_\sigma + 4n_\sigma) + r \cdot n_\Lambda^3 \cdot \text{mult}(n_\Lambda)$	$r \cdot \Pi $
QPADL-ENS	User PSD	$(r+b) \cdot ((\ell-1) \cdot t_\oplus) + n \cdot t_\oplus$	$(r+b) \cdot \ell + O(\text{polylog}(\log(n_l)))$
QPADL-FTR	User PSD	$\ell(\ell-1)rt_\oplus + 3\ell(\ell+1)t_\oplus + (n/w) \cdot t_\oplus$	$r \cdot w \cdot \ell + k \cdot b + O(\text{polylog}(\log(n_l)))$
QPADL-OOP	User PSD	$\sqrt{n} \cdot (r \cdot \ell + 1 + 1/\ell) \cdot t_\oplus + (n/2\ell)(1+r-1) \cdot t_\oplus$	$\ell \left(2\sqrt{n/(8\ell)} + \kappa/8 \right) + O(\text{polylog}(\log(n_l)))$
Service Request			
QPADL-HCT ($n_l = 2$)	User Server	$\frac{[\log_2(n_l)] \cdot 2^\kappa}{[\log_2(n_l)] \cdot \bar{H} + \bar{O}(n_\sigma \log n_\sigma + 4n_\sigma) + O(\log(n_l) \cdot \log \log(n_l)) \cdot H'}$	$\lambda + 32 + [\log_2(n_l)] \cdot H + O(\text{polylog}(\log(n_l)))$
QPADL-LBP	User Server	$\frac{O(2^{0.2925n_\Lambda + o(n_\Lambda)})}{\bar{O}(n_\Lambda^2 \cdot \text{mult}(\bar{n}_\Lambda)) + \bar{O}(n_\sigma \log n_\sigma + 4n_\sigma) + O(\log(n_l) \cdot \log \log(n_l)) \cdot H'}$	$10n_\Lambda^2 + n_\Lambda(n_\Lambda-1) + 10n_\Lambda^2 + n_\Lambda \log_2(\ \mathbf{v}\) + O(\text{polylog}(\log(n_l)))$

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_{AP} the number of APs in the region, n_Λ the lattice dimension, and n_σ the lattice dimension in ML-DSA. $\|v\|$ 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. $\text{mult}(n_\Lambda)$ refers to multiplying two n_Λ -bit numbers. Finally, κ indicates the puzzle difficulty level in the quantum setting.

Table 5

Empirical computational performance of QPADL.

Phase	Entity	Empirical Computational Cost (ms) [Client Intel (Arm)]											
PoL QPADL Intel: 4.26% Arm: 3.94%	User	33.34 (54.42)											
	AP	49.37											
Spectrum Query		$ \text{DB} = 2^{12}$ $\bar{q} = 1$ $\bar{q} = 2^7$ $\bar{q} = 2^{10}$			$ \text{DB} = 2^{14}$ $\bar{q} = 1$ $\bar{q} = 2^7$ $\bar{q} = 2^{10}$			$ \text{DB} = 2^{16}$ $\bar{q} = 1$ $\bar{q} = 2^7$ $\bar{q} = 2^{10}$			$ \text{DB} = 2^{18}$ $\bar{q} = 1$ $\bar{q} = 2^7$ $\bar{q} = 2^{10}$		
Puzzle.Bind ($\kappa = 20$)	PSD-LBP (CPU)	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 Intel: 0.6%-2% Arm: 0.3%-0.9%	User	0.258 (0.546)			0.260 (0.589)			0.273 (0.641)			0.294 (0.775)		
	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 Intel: 1.7%-2.8% Arm: 0.8%-1.2%	User	3.62 (9.01)			6.31 (15.71)			7.51 (19.09)			8.65 (21.71)		
	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 Intel: 1.3%-2.3% Arm: 0.6%-1.4%	User	2.02 (5.09)			2.72 (6.85)			5.30 (13.87)			9.43 (23.85)		
	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	38.94 (98.76)			66.36 (689.68)			316.56 (3010.76)			6251.37 (19214.14)		
	Server 0.4%	10.53											
QPADL-LBP	User	133.08 (798.48)			259.15 (1554.24)			881.41 (5288.46)			2931.45 (17588.7)		
	Server 0.3%-0.9%	9.331			10.622			11.304			11.478		

All computation costs are reported in *ms*. For client operations, we report both evaluations: results on Intel commodity hardware, with results for low-end ARM hardware shown in parentheses. Relative standard deviations are reported in the first and second column of the table for both commodity and low-end platforms. We set classical security at 128 bits per NIST guidelines and PQ security to NIST Level I, equivalent to 128-bit classical strength [8], with all parameters aligned accordingly. HCT uses SHA-256, and the LRS employs the Rescue-Prime hash function [51] with a SoK based on ethSTARK [53].

lattice-based puzzles maintain practical solving times (3 vs. 6 s) compared to hash-based ones while mitigating DoS. Moreover, in Table 6, we report the measured performance of the HCT and LBP puzzles over 1000 independent runs,

presenting the mean solving time and the corresponding mean $\pm$ standard deviation interval.

(v) *Energy Consumption on Low-Power Clients:* We estimate per-operation energy on the Cortex-A72 (Raspberry Pi 4 Model B) client as $E = P \cdot t$, adopting $P \approx$

Table 6

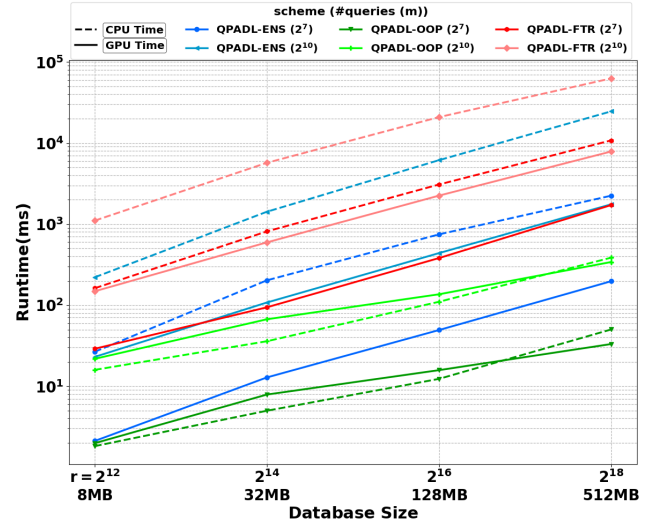
Client-side puzzle-solving cost (in seconds, standard deviation in %), over 1000 independent runs per configuration.

κ	<i>HCT</i>		<i>LBP</i>	
	ARM	Intel	ARM	Intel
14	0.096-0.101 2.1%	0.038-0.040 1.9%	0.781-0.820 2.2%	0.130-0.135 1.8%
18	0.658-0.716 4.2%	0.064-0.069 3.8%	1.50-1.63 4.1%	0.247-0.270 3.9%
20	2.89-3.15 4.3%	0.303-0.330 3.9%	5.10-5.54 4.2%	0.843-0.917 4.0%
23	17.83-20.90 8.0%	5.76-6.65 7.1%	16.48-19.09 7.4%	2.72-3.12 6.6%

5 W to reflect active single-threaded computation, which draws 2.7 W idle and up to 6.4 W under full load [47, 29]. PoL costs ≈ 273 mJ per session, and the spectrum query scales with DB size and PIR choice: across 2^{12} - 2^{18} entries, QPADL-ENS uses ≈ 2.7 -4.0 mJ, QPADL-OOP ≈ 25 -119 mJ, and QPADL-FTR ≈ 45 -108 mJ. The service-request phase is puzzle-dominated, spanning ≈ 0.5 J at the lightest setting to hundreds of joules at the heaviest, so κ and n_Λ tune per-client energy while server-side verification remains fixed.

7.2.2. Parallelization Assessment with GPU

Figure 3 shows the PSD side runtime of QPADL-ENS, QPADL-FTR, and QPADL-OOP 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 2.92 $\times$ and 19.35 $\times$, respectively. 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 GPU implementation. 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. Server-side cryptographic operations can be accelerated through GPU-based batch verification. ML-DSA verification achieves a 1.5-2 $\times$ speedup for batches of 8K or more using 32 threads per block, while Hashcash-tree verification reaches approximately 2 $\times$ speedup across nearly all security levels for batches of at least 16K. For lattice-based puzzles, larger proof sizes and host-to-device communication overhead, particularly at higher κ , limit the computation-only speedup to approximately 1.5-2 $\times$ for


Figure 3: Scalability Benchmarking of PIRs on CPU/GPU

batches of 8K or more. Overall, GPU acceleration improves end-to-end cryptographic processing by up to 2 $\times$.

7.2.3. Communication Overhead

(i) The location commitment includes the location coordinates (16 bytes), beacon (8 bytes), timestamp (8 bytes), and a random nonce (16 bytes), and the PoL request additionally carries a 32-byte window-scoped pseudonym transmitted only to the AP. The ring signature size scales with the number of ring members, measuring approximately 18 KB for 2^3 users and 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 [54]. Although ML-KEM is faster than RSA (used in Tor), its use still requires two packet transmissions due to Tor's 512-byte packet size, resulting in an average circuit build time of 300 ms. As each retrieved block is 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.

7.2.4. Proximity Verification Assessment

We evaluate ProxVerif through Monte Carlo analysis [46] under a log-distance path-loss channel model

with log-normal shadowing, rather than packet-level network simulation, since proximity-verification accuracy is governed by ranging error and the RSS/RTT trade-off rather than network-layer dynamics.

Threshold Calibration and Detection Accuracy. Figure 4 places legitimate clients uniformly within the AP coverage disk and spoofers outside it. Panel (a) shows legitimate and spoofer acceptance as a function of Δ_{th} : as the threshold loosens, legitimate acceptance rises first, since legitimate clients lie within the coverage radius, whereas spoofer acceptance rises later, since spoofers lie beyond it. This separation yields an operating point near $\Delta_{th} \approx 1.1 r_{AP}$ that admits 88% of legitimate clients while accepting only 11% of spoofers, consistent with the threshold-selection rule of Section 4.1. This residual acceptance reflects ranging noise near the coverage boundary; it is not the system's spoofing bound, since an admitted spoofer still obtains only a region-scoped grant whose channel and power parameters are unusable outside that region. Operators may tighten Δ_{th} to trade legitimate acceptance for lower spoofer admission. Panel (b) reports spoofer acceptance as a function of the attacker's absolute distance from the AP, for coverage radii of 30, 50, and 100 m. Attack success is highest when the spoofer sits at the coverage boundary ($\approx r_{AP}$), where it lies only marginally beyond the threshold and ranging noise occasionally admits it, but falls below 15% at $1.5 r_{AP}$ and below 3% at $2 r_{AP}$; thus distant spoofers are reliably rejected, and this behavior is consistent across AP densities.

Spoofing-Resistance Evaluation. Among the location-spoofing threats in our model (Section 3.1), we distinguish three cases by how each is countered. *Proof replay and transfer*, i.e., reusing or forwarding a previously issued artifact, are prevented cryptographically by the event-scoped, per-window binding of the proof artifact and its linkability (Definition 8), independently of the physical channel. *Physical distance manipulation requiring light-speed-bounded timing* lies outside our proximity model. The remaining and most relevant case for ProxVerif is *relay-based distance hijacking*, in which a distant malicious client relays its messages through a compromised benign client legitimately within proximity, thereby borrowing the relay's RSS. The evaluation uses a log-distance path-loss channel (exponent 3.0) with log-normal shadowing (4 dB standard deviation) for RSS ranging and additive Gaussian jitter (3 m) for RTT ranging; the swept attacker and configuration parameters are summarized in Table 3.

Because the relay hop introduces additional propagation delay, the RTT continues to reflect the malicious client's true, larger distance, even though the borrowed RSS suggests proximity. ProxVerif fuses the two estimates as $\Delta_c = w_{RTT} \cdot d_{RTT} + (1 - w_{RTT}) \cdot d_{RSS}$ and accepts when $\Delta_c \leq \Delta_{th}$. Figure 5 reports the attack success rate over 100 trials per configuration, sweeping the relay distance, the malicious-client distance, and the fusion weight w_{RTT} . The attack succeeds only when RSS dominates the decision and the relay is close to the AP (up to 73% for $w_{RTT} \leq 0.2$), and collapses to 2.3% once RTT is weighted more heavily ($w_{RTT} \geq 0.8$), since the relay cannot reduce round-trip time below the physical propagation limit. This confirms that

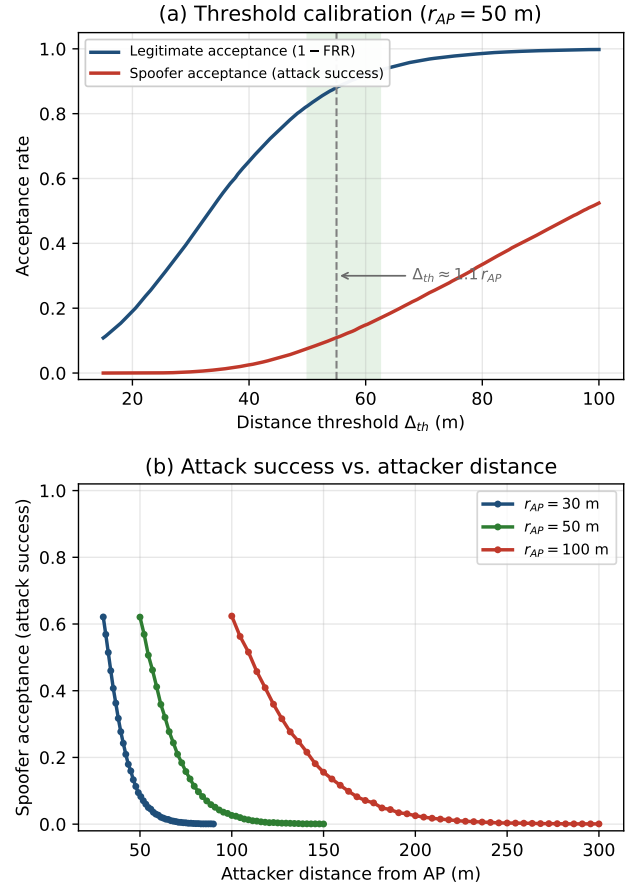


Figure 4: ProxVerif threshold calibration and detection accuracy under a log-distance path-loss channel model. (a) Legitimate versus spoofer acceptance as a function of the distance threshold Δ_{th} ; the shaded band marks the operating point near $\Delta_{th} \approx 1.1 r_{AP}$, where legitimate acceptance is high while spoofer acceptance remains low. (b) Spoofer acceptance versus the attacker's absolute distance from the AP, for radii of 30, 50, and 100 m; attack success decays sharply as the attacker moves beyond the coverage boundary.

ProxVerif resists relay-based distance hijacking when configured to emphasize RTT, and quantifies the RSS/RTT weighting that governs this resistance.

7.2.5. 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. Figure 6 shows communication domination for a few SUs, whereas with more SUs and larger DB sizes, computation becomes the bottleneck. GPU acceleration overcomes this: QPADL-ENS achieves up to 4.40 $\times$, QPADL-FTR up to 11.51 $\times$, and QPADL-OOP 1.91 $\times$ speedup for up to 2^{10} SUs per grid and time window. Although GPU-accelerated QPADL-OOP achieves the best performance, its speedup is smaller due to lower CPU overhead, while the gains for QPADL-ENS and QPADL-FTR grow with larger query volumes and DB sizes.

Scaling in the number of PSDs is characterized analytically in Table 4: per-PSD response cost is independent of ℓ , since each server computes over its own replica,

Distance-hijacking attack success against ProxVerif

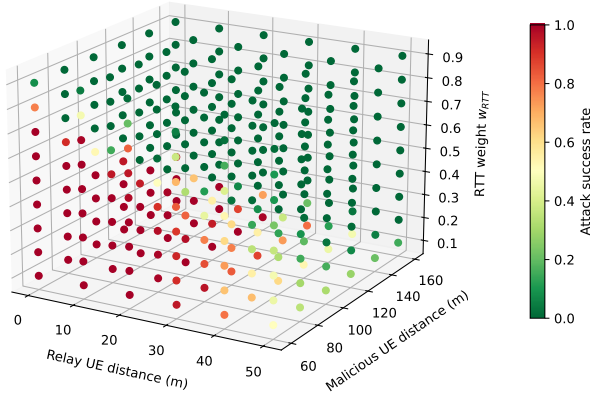


Figure 5: Attack success rate of relay-based distance hijacking against ProxVerif, over relay-UE distance, malicious-UE distance, and the RTT fusion weight w_{RTT} . Warmer colors indicate higher success; emphasizing RTT ($w_{RTT} \geq 0.8$) reduces success to 2.3%.

whereas client-side reconstruction and total communication grow linearly in ℓ (quadratically for the Byzantine-robust QPADL-FTR). Our experiments fix $\ell = 3$ because ℓ is bounded by deployment rather than by performance: the FCC certifies a small number of SAS administrators, and the non-collusion assumption underlying multi-server PIR is credible only across independently operated administrators, so ℓ remains small and does not vary with deployment scale.

Scaling to Metro-Scale Deployments. The 2^{10} SUs per grid and time window used in our evaluation is a per-grid workload rather than a system-wide limit. SAS is inherently spatially partitioned: each client queries the DB block indexed by its own grid cell (l_x, l_y) , and spectrum decisions, PoL, and rate-limiting are all resolved locally within a cell and its serving APs. Consequently, a metro-scale deployment comprising 10^4 - 10^5 CBSDs corresponds not to a single query pool but to many independent grids, each bounded by the per-grid costs reported above. Because queries in distinct grids share no data dependency, they parallelize across PSD instances, so metro-scale load is absorbed by horizontal replication of PSD capacity rather than by increased per-query work. The two dimensions that do grow with deployment size (i.e., DB size and AP ring size) are precisely those characterized in our experiments: Figure 3 and Figure 6 sweep DB size to 2^{18} entries (512 MB) and show that GPU acceleration and compression keep per-query PSD cost tractable at that scale, while LRS signing grows only mildly with ring size (20 ms at 2^3 to 60 ms at 2^{13} AP members). QPADL therefore scales to dense vehicular and metro rollouts through per-grid locality and inter-grid parallelism, with the reported per-grid figures serving as the extrapolation unit.

7.2.6. DoS Resilience Under Adversarial Load

Beyond the per-request puzzle cost, we evaluate QPADL's resilience when the service server is actively flooded. We model the server as a bounded worker-pool queue and

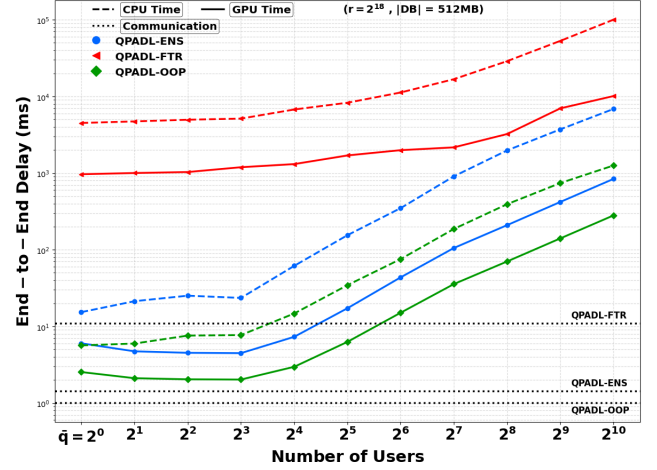


Figure 6: End-to-End Cryptographic Delay

Table 7

Parameters of the DoS resilience evaluation.

Parameter	Value
Server workers	32
Queue capacity	200
Per-attacker rate, no counter-DoS	50 req/s (network-bound)
Honest processing cost	10.53/11.30 ms (HCT/LBP)
Batched rejection cost (GPU, 32K)	0.0038/0.0040 ms/req
Per-attacker valid-token rate ($\kappa=20$)	3.16/1.14 req/s (HCT/LBP)
Concurrent attackers	0-1600
Puzzle difficulty κ	14-23

simulate request admission through a discrete-event queue model, rather than packet-level network simulation, since DoS resilience is governed by queue-saturation dynamics; all costs are taken from our measurements (Tables 5 and 7).

Attack strategies. A malicious service request is rejected at one of three stages: (i) *random-garbage* floods fail ML-DSA signature verification; (ii) *invalid-solution* floods pass the signature check but fail PoW verification; and (iii) *replayed* tokens are caught by event-scoped linkability. All three are shed after a single, cheap verification. Producing a *valid* token instead requires solving a puzzle, which rate-limits the attacker. We evaluate the two extremes: the cheapest-to-reject invalid floods, and the strongest *valid-token* flood in which the adversary genuinely solves puzzles. In the undefended baseline the attacker faces no computational obligation, so its rate is network-bound; we set it to 50 requests/s per attacker, against 3.16 (HCT) and 1.14 (LBP) valid requests/s under QPADL at $\kappa = 20$. The gap between these rates, rather than any assumed server fragility, is what produces the throughput separation in Figure 8.

Cost asymmetry and instantiation trade-off. Figure 7(a) contrasts the attacker's puzzle-solving cost with the server's per-request verification cost. Solving a puzzle costs 38.9-6251 ms (HCT) or 133-2931 ms (LBP) depending on κ , whereas server-side verification is a flat ≈ 10.5 -11.3 ms, yielding a $30\times$ (HCT) to $78\times$ (LBP) cost asymmetry at $\kappa = 20$ that widens with κ . This asymmetry is the core of the defense: the adversary must expend orders of magnitude more computation to produce a request than the server

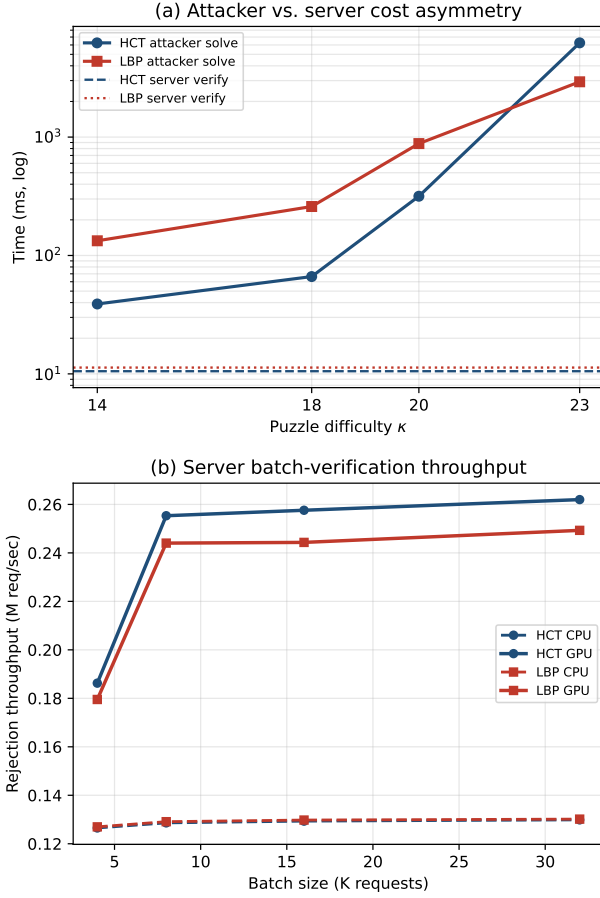


Figure 7: DoS cost structure of QPADL. (a) Attacker puzzle-solving cost versus server per-request verification cost across difficulty κ , for HCT and LBP; the gap is the 30-78 $\times$ cost asymmetry underlying the defense. (b) Server-side rejection throughput under CPU and measured GPU batch verification of concurrent requests, showing up to a 2 $\times$ improvement.

spends to process or reject it. Figure 7(b) shows that GPU batch verification of independent concurrent requests further raises the server's rejection throughput by up to 2.0 $\times$ over CPU (from 0.13 to 0.26 M requests/sec at a 32K batch), lowering the server's amortized cost to shed floods without reducing the attacker's puzzle cost. We emphasize that batching processes many independent requests in parallel for an S -fold throughput gain; it does not verify a batch at the cost of a single request.

Honest service under flood. Figure 8 reports honest-user service delay and throughput under a valid-token flood, the strongest strategy, in which attackers solve puzzles so their requests pass verification but remain rate-limited by the puzzle cost. Without a counter-DoS mechanism, honest throughput collapses below 4% and waiting time saturates at ≈ 65 ms. With QPADL, honest service is preserved far longer: QPADL-HCT sustains full throughput up to ≈ 400 concurrent attackers and degrades gracefully thereafter (54% at 1600), while QPADL-LBP maintains near-full throughput across the entire range, since its heavier puzzle throttles each attacker to ≈ 1.1 valid requests/sec

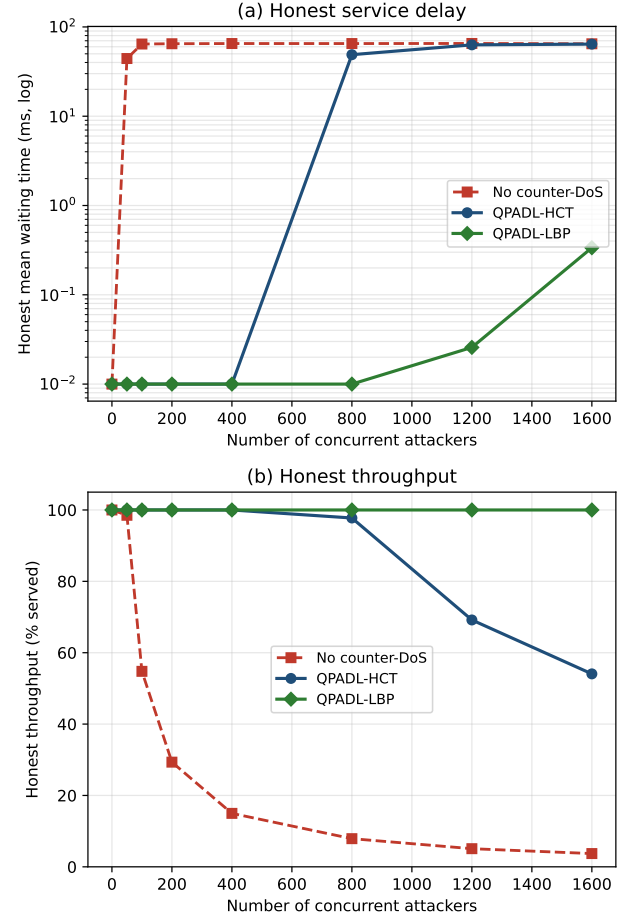


Figure 8: Honest-user service under a valid-token flood: (a) mean waiting time and (b) throughput, versus the number of concurrent attackers. An undefended server collapses, whereas QPADL-LBP sustains near-full service across the range and QPADL-HCT degrades gracefully at extreme scale, reflecting the instantiation trade-off. Server-side rejection of invalid and replayed floods is instantiation-independent; see Figure 7 for the HCT/LBP and CPU/GPU distinctions.

versus HCT's ≈ 3.2 . This quantifies the instantiation trade-off: LBP offers stronger DoS resilience at higher communication overhead, whereas HCT is lighter but admits a higher adaptive-attacker rate. Invalid-solution and replay floods are shed even more cheaply at batched-verification cost and do not degrade honest service, and malicious rejection accuracy remains at $\approx 100\%$ throughput.

Relevance to SAS deployment. These results are well matched to the operational scale of SAS/CBRS. Spectrum access is spatially partitioned: a given service server handles requests from clients within a bounded set of grids and time windows rather than a global population, so the concurrent-attacker counts at which QPADL sustains full service ($\sim 10^2$ - 10^3) comfortably cover realistic per-server adversarial loads. Moreover, QPADL's defense is *proactive* rather than *reactive*: the puzzle requirement and event-scoped rate limiting are enforced at admission, before a request consumes server resources, so the server never needs to detect an attack in progress or provision for worst-case reactive filtering. Combined with GPU-accelerated batch

rejection and the operator's freedom to raise κ or select the LBP instantiation under elevated threat, QPADL provides a tunable, deployment-ready DoS posture for FCC-compliant spectrum access.

Additional adversarial considerations. Replayed tokens are caught by the one-use-per-window linkability of the proof artifact and rejected after a single verification. Spreading a flood across time windows does not help, as each window requires a freshly solved puzzle (Figure 7(a)). Spreading across APs does not by itself help: e_{ID} is derived from the region-scoped ring and beacon together with the client's window-scoped pseudonym, none of which are AP-specific, so all artifacts carrying the same pseudonym collide in the admission log within a window and only the first is served, whichever AP issued them. An adversary with (C1) evades this by presenting a fresh pseudonym on a fresh radio-layer identity, which requires physical presence within coverage and yields one additional admitted query per identity. This bounds query-side abuse at the rate at which the adversary can churn identities under live proximity verification; it does not affect service-side resilience, where each admitted request still carries its own puzzle cost. PSD and AP load are bounded by the measured PIR response and proximity-check costs together with per-window admission, and low-power honest clients are unaffected by the flood since honest service is preserved regardless of attack intensity, their own puzzle cost being a one-time, κ -tunable parameter.

7.2.7. Comparison with SOTA

Existing PIR-based schemes, such as [33, 58, 23], mainly protect the queried index, but they do not jointly provide strong communication anonymity, PQ security, and integrated service-side abuse resistance. From a practical perspective, QPADL also remains scalable: GPU acceleration yields 4.40 $\times$ speedup for QPADL-ENS, 11.51 $\times$ for QPADL-FTR, and 1.91 $\times$ for QPADL-OOP for up to 2^{10} SUs per grid and time window, showing that the framework remains deployable even as the query load grows. Moreover, in the representative QPADL-OOP instantiation, the PSD-side online cost remains as low as 110.84 ms for $|DB| = 2^{16}$ and $\bar{q} = 2^{10}$ under GPU execution, compared with 323.85 ms for QPADL-ENS and 2144.63 ms for QPADL-FTR under the same setting.

From the location-assurance perspective, only a limited subset of prior works incorporates explicit PoL or location-verification support, and these typically rely on trusted auxiliary entities, classical assumptions, or significantly higher latency. For example, Li et al. [43] and Xin et al. [58] report PoL costs of about 210 ms and 430.1 ms, respectively, whereas QPADL achieves PQ-secure proximity-backed proof generation with only 33.34 ms on the client side and 49.37 ms on the AP side. This yields a combined cryptographic PoL cost of about 82.71 ms, while additionally providing event-scoped misuse detection through the LRS-based proof artifact. Hence, unlike earlier query-private SAS schemes that largely assume honest users, QPADL directly accounts for replay, spoofing-oriented misuse, and proof forgery within the access workflow itself.

All schemes are contrasted under a common system model, security level, and database configuration; communication overheads are platform-independent and directly comparable, while computational figures for prior schemes are as reported in their papers, which if anything understates QPADL's margin given the older platforms involved.

From the communication and DoS-resilience perspectives, prior solutions usually optimize one dimension at the expense of the others. Existing privacy-preserving schemes incur substantial communication overhead, e.g., about 753 KB for LP-Chor [31], 6 MB for LP-Goldberg [31], 125 KB for RAID-LP-Chor [31], 1.25 MB for TrustSAS [33], and 605.92 KB for PACDoSQ [23], while Xin et al. [58] still requires 325 KB in a single-DB setting. In contrast, each retrieved block in QPADL remains below 50 KB, and the PQ-Tor communication delay is bounded at about 175 ms, with an average circuit-build time of about 300 ms. At the service layer, QPADL further embeds authenticated computational throttling directly into the workflow: HCT solving costs range from 38.94 ms to 316.56 ms for moderate difficulty levels, while the lattice-based alternative provides PQ-secure throttling with client-side costs from 133.08 ms to 881.41 ms. Therefore, unlike detection-only defenses or isolated request-throttling methods, QPADL offers a more complete and deployment-oriented security architecture for SAS operation during the PQ transition.

8. Conclusion and Future Work

This work introduced QPADL, the first framework to simultaneously address location privacy, anonymity, proximity-backed location assurance against 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. Future work includes extending QPADL to user mobility and handover scenarios, and incorporating hardware-assisted distance bounding for coordinate-level location verification.

A. APPENDIX

A.1. Security Analysis

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

Theorem 1. *For any QPT adversary interacting with QPADL, the framework achieves, except with probability $\text{negl}(\lambda)$: (i) PQ-secure location private spectrum access, where the privacy of the queried spectrum index reduces to the security of the instantiated PIR layer, namely IT-secure*

for QPADL-ENS and QPADL-FTR, and computationally-secure for QPADL-OOP under the hardness of its underlying PRG and preprocessing model. (ii) PQ communication anonymity and unlinkability, where the confidentiality of the anonymous transport reduces to the IND-CPA security of the layered encryption and the PQ security of the KEM, relying on standard symmetric-key assumptions together with Module-LWE-based hardness. (iii) Authenticity and pseudonym-scoped event linkability of the proof artifact, where the validity and misuse-detection properties of the proximity-backed proof artifact reduce to the simulation-extractability and soundness of the underlying SoK together with the one-wayness, collision resistance, and second-preimage resistance of the employed hash functions, applied to the event-identifier derivation. (iv) Authentic and abuse-resistant service access, where puzzle authenticity reduces to the EUF-CMA security of ML-DSA under standard module-lattice assumptions (e.g., Module-SIS/Module-LWE), while service-side computational throttling and replay/rate-limit enforcement reduce to the hardness of the selected puzzle construction, the hash-based binding of tokens, commitments, and event identifiers, and the collision resistance of H and H' underlying event-identifier uniqueness. Rate-limit enforcement is per pseudonym, and per client against an adversary lacking (C1) (Section 3.1).

Proof. (i) *t-private PQ-secure Location Privacy:* QPADL is instantiated with PIR schemes that ensure t -private location privacy during spectrum access, meaning that any coalition of at most t corrupted PSDs learns nothing about the queried index θ beyond negligible probability, as formalized in Definition 4. In particular, both QPADL-ENS and QPADL-FTR provide information-theoretic privacy: QPADL-ENS achieves privacy against up to $\ell - 1$ colluding servers, while QPADL-FTR achieves t -private retrieval together with its robustness guarantees. Hence, for any QPT adversary $\mathcal{A}$ corrupting a subset $S \subseteq [\ell]$ with $|S| \leq t$ and for any two query indices $\theta_0, \theta_1 \in \text{DB}$, $\left| \Pr[\mathcal{A}(\text{View}_S^{\text{PIR}}(\theta_0)) = 1] - \Pr[\mathcal{A}(\text{View}_S^{\text{PIR}}(\theta_1)) = 1] \right| \leq \text{negl}(\lambda)$. For the information-theoretic instantiations, this bound is independent of the adversary's computational power, including quantum capabilities. In contrast, the QPADL-OOP provides computational PIR privacy, where any non-negligible distinguishing advantage against the queried index yields an adversary against the hardness of the underlying PRG and preprocessing model, as formalized in Lemmas 1 and 2.

(ii) *PQ Communication Anonymity and Unlinkability:* QPADL preserves client anonymity and unlinkability against PSDs and external observers through PQ-Tor with three relays (N_e, N_m, N_x), where each relay learns only its predecessor and successor, as formalized in Definition 7. Messages are transmitted over an onion circuit with layered symmetric encryption, $\text{ctxt} = \text{Enc}_{sk_{N_e}}(\text{Enc}_{sk_{N_m}}(\text{Enc}_{sk_{N_x}}(m)))$, where the session keys $sk_{N_i} \in \{0, 1\}^{256}$ are established via PQ-secure KEM encapsulation and decapsulation, $\text{ctxt}'_i \leftarrow \text{ML-KEM.Encaps}(PK_{N_i, \text{Kyber}}, sk_{N_i} \leftarrow \text{ML-KEM.Decaps}(sk_{N_i, \text{Kyber}}, \text{ctxt}'_i)$, for $i \in \{e, m, x\}$.

Hence, for any QPT adversary $\mathcal{A}$ attempting to distinguish between two executions that differ only in the honest sender or receiver, its advantage is bounded by the IND-CPA security of the layered encryption and the Module-LWE-based hardness underlying the KEM, i.e., $\text{Adv}_{\mathcal{A}}^{\text{PQ-Tor-Anon}}(\lambda) \leq \text{Adv}_{B_1}^{\text{IND-CPA}}(\lambda) + \text{Adv}_{B_2}^{\text{MLWE}}(\lambda) + \text{negl}(\lambda)$. Thus, any non-negligible break of communication anonymity or unlinkability in QPADL yields an adversary against either the confidentiality of the layered encryption or the PQ security of the underlying KEM. Concretely, this relies on AES-256 offering about 128-bit post-quantum security under Grover's algorithm [10], together with the security of Module-LWE-based KEMs, whose hardness is related to worst-case module-lattice problems such as MSVP in the random oracle model [11].

(iii) *Proximity-Backed Location Assurance and Spoofing Resistance:* In QPADL, proximity-backed location assurance relies on the authenticity and event-scoped linkability of its PQ LRS scheme [59], 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 $R = \{PK_1, \dots, PK_n\}$, where $n = 2^k$ and $k = \lceil \log_2(n) \rceil$. The relation proven in the SoK is formalized as $R_s = \{((e, rt, T), (P, l, sk_l)) : T = H'(sk_l, e), rt = \text{MPath}(H'(sk_l), P, l)\}$, where $e \in \mathbb{F}_p$ is the event identifier, T is the linkability tag, rt is the Merkle root, l is the binary index of PK_l in the tree, and P is the corresponding Merkle path. For any two signatures σ_1, σ_2 on messages m_1, m_2 , event-scoped linkability holds whenever their tags satisfy $H'(sk_l, e_1) = H'(sk_l, e_2)$. Thus, reuse of a proof artifact under the same event scope is detectable. Linkability in QPADL is pseudonym-specific rather than AP-specific. The event identifier is derived as $e_{\text{ID}} = H'(R_{\text{AP}}, \beta_{\text{TW}}, \text{TW}, \text{nym}_{\text{TW}})$ with $\text{nym}_{\text{TW}} = H'(sk_c, \text{TW})$ contributed by the client, while R_{AP} and β_{TW} are region-scoped and carry no AP-specific component. Since nym_{TW} is deterministic in the client's registration secret and the active window, any two artifacts obtained under the same pseudonym within TW, from any AP of the region, induce an identical e_{ID} and are therefore detected at admission by the per-window log $\mathcal{L}_{\text{TW}}$; a fresh commitment nonce does not evade this, as r_c does not enter the derivation of e_{ID} . Detection is thus per pseudonym unconditionally, and per client for any adversary lacking (C1). When both artifacts are issued by the same AP, the tags $T = H'(sk_{\text{AP}}, e_{\text{ID}})$ additionally coincide and reuse is detected by LRS.Link; across distinct APs the tags differ, which is why enforcement is anchored at the event scope rather than at the tag. Conversely, distinct clients yield $\text{nym}_{\text{TW}} \neq \text{nym}'_{\text{TW}}$, and hence $e_{\text{ID}} \neq e'_{\text{ID}}$, except with probability negligible in λ under the collision resistance of H' , so artifacts issued to different clients within the same window remain unlinked and honest requests are not falsely rejected. Because TW enters the derivation, pseudonyms and event

identifiers across distinct windows are unlinkable under the one-wayness of H' , preserving cross-session anonymity while supporting one-use-per-window rate limiting. The residual linkage is therefore exactly the intended one: within a single window and region, a PSD can recognize repeated requests bearing the same e_{ID} without learning $n_{ym_{TW}}$, sk_c , or the client's coordinates. The zero-knowledge property of the NIAoK ensures that no information beyond validity leaks, while unforgeability reduces to the collision resistance and preimage resistance of H' , together with the non-slanderability and extractability of the underlying SoK ([59] for details). Under the structured-hash assumptions and the one-wayness of $MPATH$, any adversary $\mathcal{A}$ attempting to forge a valid proof artifact or violate event-scoped linkability has success probability bounded by $Adv_{\mathcal{A}}^{\text{forge}}(\lambda) \leq Adv_{\mathcal{A}'}^{\text{SoK}}(\lambda) + Adv_{\mathcal{A}''}^{\text{Hash}}(\lambda) \leq \text{negl}(\lambda)$, where $\mathcal{A}'$ breaks SoK extractability and $\mathcal{A}''$ finds a hash collision or preimage. Condition (S) of Definition 8 follows from the unforgeability of the LRS artifact together with the AP's ProxVerif check, which no ring member signs without; condition (F) follows from the binding of TW and β_{TW} into e_{ID} and from one-use-per-window admission at the PSDs. Therefore, within the assumed proximity-verification model of QPADL, the resulting proof artifact is authentic, unlinkability-preserving across different event scopes, and misuse-detectable within the same event scope, thereby providing cryptographic protection against proof forgery, replay, and related spoofing attempts without claiming full physical-layer secure localization (cf. Definition 8).

(iv) *Authentic and Abuse-Resistant Service Access:* In QPADL, puzzle authenticity is enforced through the existential unforgeability of ML-DSA 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 by $Adv_{\mathcal{A}}^{\text{EUF-CMA}}(\lambda) := \Pr[(\Pi_\theta^*, \sigma_\theta^*) \leftarrow \mathcal{A} \text{ and } (\Pi_\theta^*, \sigma_\theta^*) \notin \mathcal{Q}] \leq \text{negl}(\lambda)$, where $\mathcal{Q}$ is the set of honestly issued puzzle-signature pairs. This guarantee relies on the standard module-lattice assumptions underlying ML-DSA, namely Module-LWE and Module-SIS hardness, with security reducing tightly to MSIS in the quantum ROM; hence, only puzzles issued by an authorized PSD are accepted except with negligible probability.

The service-side anti-abuse mechanism in QPADL is instantiated through CPPs, relying either on the second-preimage resistance of the underlying hash function in QPADL-HCT or on the hardness of the Hermite-SVP problem in QPADL-LBP, as formalized in Lemmas 3 and 4, respectively. In addition, per-pseudonym rate-limiting at PSDs relies on the event-scoped binding of commitments and the linkability of the proof artifact. Specifically, the committed value is of the form $C_{TW} = H((l_x, l_y) \parallel \beta_{TW} \parallel TW \parallel r_c)$, where the location, beacon, active time window, and nonce are cryptographically bound by the one-wayness, collision resistance, and second-preimage resistance of H . Hiding follows from the λ -bit nonce: for any two candidate coordinate pairs, the induced commitment distributions are indistinguishable in the ROM, so neither the AP nor the

SAS server learns (l_x, l_y) from C_{TW} despite β_{TW} and TW being public. An adversary $\mathcal{A}$ attempting to bypass replay or one-use-per-window enforcement must either produce a fresh valid proof artifact for the same event scope without the required witness, break event-scoped linkability, or find distinct openings that yield the same commitment, i.e., $H((l_x, l_y) \parallel \beta_{TW} \parallel TW \parallel r) = H((l'_x, l'_y) \parallel \beta'_{TW} \parallel TW' \parallel r')$, for two distinct tuples. Thus, any successful replay, token misuse, or rate-limit bypass implies either a break of the proof-artifact assumptions in part (iii), a violation of the binding properties of H , or a break of the selected CPP hardness assumption. Therefore, QPADL provides authenticated puzzle issuance, per-request computational throttling, and abuse-resistant service access under the stated assumptions. $\square$

Lemma 1. *QPADL-ENS and QPADL-FTR achieve IT-secure private spectrum retrieval, where QPADL-ENS guarantees $(\ell-1)$ -private query-index privacy and QPADL-FTR guarantees t -private query-index privacy together with v -Byzantine robustness and k -out-of- ℓ correctness under the Shamir-secret-sharing and decoding conditions.*

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 $C \subset \{1, \dots, \ell\}$ 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 PSD_j the query $\rho_j := \langle f_1(\alpha_j), \dots, f_r(\alpha_j) \rangle$, receiving response $R_j := \rho_j \cdot \text{DB}$. 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 $v < k - \lfloor \sqrt{kt} \rfloor$ Byzantine responses when $k > t$ servers reply under (ℓ, t) -Shamir secret sharing. $\square$

Lemma 2. *QPADL-OOP achieves computationally secure private spectrum retrieval against polynomial-time adversaries, where query-index privacy reduces to the pseudorandomness of the underlying PRG together with the preprocessing and chunk-permutation assumptions of the construction.*

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 $R_i := A_i \oplus q_i \cdot \text{chunk}_0$, where $A := q \cdot \text{DB}$ and $q := \text{PRG}(S_i, k(t-1))$, as shown in Fig. 2. 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: $|\Pr[\mathcal{A}(\{\rho_i^{(j)}\}_{i \in C}) \mid \theta = \theta'] - \Pr[\mathcal{A}(\{\rho_i^{(j)}\}_{i \in C}) \mid \theta = \theta'']| \leq \text{negl}(\lambda)$, where the distinguishing advantage is bounded as: $\text{Adv}_{\mathcal{A}}^{\text{Privacy}}(\lambda) := \max_{\theta', \theta''} |\Pr[\mathcal{A}(\text{view}_{\theta'})] - \Pr[\mathcal{A}(\text{view}_{\theta''})]| \leq \text{Adv}_{\mathcal{A}'}^{\text{PRG}}(\lambda)$ with $\mathcal{A}'$ being a reduction that breaks the PRG's PQ security. $\square$

Lemma 3. *QPADL-HCT provides hash-based computational throttling for service access, where the adversary's success probability in bypassing the required client work reduces to the second-preimage resistance of the underlying hash function and the sequential structure of the HCT.*

Proof. In QPADL-HCT, the client solves a sequence of $\ell = \lceil \log_2(n_l) \rceil$ hashcash puzzles along a selected path in a binary hash tree with n_l leaves, where each puzzle instance is parameterized by $\pi = (h, n_s, \kappa, n_l)$. For each node on the path, the probability that a $\mathcal{A}$ finds a valid solution in one trial is $2^{-\kappa}$, so the expected work per puzzle is $O(2^\kappa)$ classically and $O(2^{\kappa/2})$ under Grover-style quantum search. Since the verifier checks a valid root commitment together with all puzzle solutions along the selected path, the total expected work per request is $\ell \cdot 2^\kappa = \lceil \log_2(n_l) \rceil \cdot 2^\kappa$ in the classical setting, with a corresponding quadratic quantum speedup in the brute-force search term only. Moreover, because each higher-level node depends on the validity of its descendants, the path must be solved in sequence; thus, the construction enforces at least $\lceil \log_2(n_l) \rceil$ sequential proof-of-work steps and cannot be fully parallelized. Any adversary attempting to bypass this work must either solve all path puzzles successfully or produce a distinct invalid path or node assignment that still verifies against the committed root, which would violate the second-preimage resistance of the underlying hash function. Therefore, the adversary's success probability is bounded by $\text{Adv}_{\mathcal{A}}^{\text{HCT}}(\lambda) \leq \ell \cdot 2^{-\kappa} + \text{Adv}_{\mathcal{B}}^{\text{2nd-Pre}}(\lambda)$, where $\ell = \lceil \log_2(n_l) \rceil$. For standard parameter choices, this quantity is negligible in λ (or in κ when κ is selected as a function of λ). Hence, QPADL-HCT provides verifiable hash-based throttling against request flooding under the second-preimage resistance of $\mathcal{H}$ and the sequential-work structure of the HCT. $\square$

Lemma 4. *QPADL-LBP provides lattice-based computational throttling for service access, where the adversary's success probability in bypassing the required client work reduces to the hardness of the underlying α -Hermite Shortest Vector Problem instance family.*

Proof. In QPADL-LBP, the lattice-based PoW instance $\pi = (\alpha, n_\Lambda, B, p)$ relies on the hardness of the α -Hermite Shortest

Vector Problem. Specifically, the challenge is to compute a nonzero vector of the lattice $\mathbf{v} \in \Lambda(B) \setminus \{0\}$ such that $\|\mathbf{v}\| \leq \alpha \cdot \lambda_1(\Lambda)$, where $\lambda_1(\Lambda) = \min\{\|\mathbf{u}\| : \mathbf{u} \in \Lambda \setminus \{0\}\} \leq p^{1/n_\Lambda}$ 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 denoting the lattice dimension. Thus, producing a valid service token requires solving an α -Hermite-SVP instance over $\Lambda(B)$. Using the best-known attacks, such as lattice reduction and enumeration, solving these puzzles requires on average $O(2^{0.2925 \times n_\Lambda + o(n_\Lambda)})$ operations in the classical setting and $O(2^{0.2570 \times n_\Lambda + o(n_\Lambda)})$ operations in the quantum setting. Hence, for any PPT or QPT adversary $\mathcal{A}$ attempting to obtain a valid token without performing the prescribed work, its success probability is bounded by the hardness of the underlying lattice problem, i.e., $\text{Adv}_{\mathcal{A}}^{\text{LBP}}(\lambda) \leq \text{Adv}_{\mathcal{B}}^{\text{Hermite-SVP}}(\lambda) + \text{negl}(\lambda)$, where $\mathcal{B}$ is a reduction that uses $\mathcal{A}$ to solve the corresponding α -Hermite-SVP instance. Equivalently, this implies $\Pr[\mathcal{A} \text{ finds } \mathbf{v} \in \Lambda(B) \text{ s.t. } \|\mathbf{v}\| \leq \alpha \cdot \lambda_1(\Lambda(B))] \leq 2^{-\kappa}$, for security parameter κ induced by the selected lattice parameters. Each puzzle remains valid only for a limited duration determined by κ , which in turn depends on α , λ , and n_Λ . Therefore, except with negligible probability, QPADL-LBP enforces verifiable client work and provides abuse-resistant service access under the hardness of the selected lattice parameters. $\square$

Acknowledgment

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

References

- [1] Agarwal, P., Manekiya, M., Ahmad, T., Yadav, A., Kumar, A., Donelli, M., Mishra, S.T., 2022. A survey on citizens broadband radio service (cbrs). *Electronics* 11, 3985.
- [2] Albrecht, M.R., Ducas, L., Herold, G., Kirshanova, E., Postlethwaite, E.W., 2019. The general sieve kernel and new records in lattice reduction, in: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer.
- [3] Ali, I.M., Caprolu, M., Pietro, R.D., 2020. Foundations, properties, and security applications of puzzles: A survey. *ACM Computing Surveys (CSUR)* 53, 1–38.
- [4] Alviano, M., 2023. Hashcash tree, a data structure to mitigate denial-of-service attacks. *Algorithms* 16, 462.
- [5] Aura, T., Nikander, P., Leiwo, J., 2000. Dos-resistant authentication with client puzzles, in: *International workshop on security protocols*, Springer. pp. 170–177.
- [6] Back, A., 2002. Hashcash: A Denial of Service Counter-Measure. Technical Report. Palo Alto, CA, USA.
- [7] Bahrak, B., Bhattarai, S., Ullah, A., Park, J.M.J., Reed, J., Gurney, D., 2014. Protecting the primary users' operational privacy in spectrum sharing, in: *2014 IEEE International Symposium on Dynamic Spectrum Access Networks (DYSpan)*, IEEE. pp. 236–247.
- [8] Bavdekar, R., Chopde, E.J., Agrawal, A., Bhatia, A., Tiwari, K., 2023. Post quantum cryptography: a review of techniques, challenges and standardizations, in: *2023 International Conference on Information Networking (ICOIN)*, IEEE. pp. 146–151.
- [9] Behnia, R., Postlethwaite, E.W., Ozmen, M.O., Yavuz, A.A., 2021. Lattice-based proof-of-work for post-quantum blockchains, in: *International Workshop on Data Privacy Management*, Springer.
- [10] Bonnetain, X., Naya-Plasencia, M., Schrottenloher, A., 2019. Quantum security analysis of aes. *IACR Transactions on Symmetric Cryptology* 2019, 55–93.

- [11] Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D., 2018. Crystals-kyber: a cca-secure module-lattice-based kem, in: 2018 IEEE European symposium on security and privacy (EuroS&P), IEEE. pp. 353–367.
- [12] Bostanov, V., 2021. Client puzzle protocols as countermeasure against automated threats to web applications. IEEE Access 9.
- [13] Caleffi, M., Cacciapuoti, A.S., 2014. Database access strategy for TV white space cognitive radio networks, in: 2014 Eleventh Annual IEEE International Conference on Sensing, Communication, and Networking Workshops (SECON Workshops), pp. 34–38.
- [14] Canetti, R., Chen, Y., Holmgren, J., Lombardi, A., Rothblum, G.N., Rothblum, R.D., Wichs, D., 2019. Fiat-shamir: from practice to theory, in: Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, pp. 1082–1090.
- [15] Chakraborty, T., Mitra, S., Mittal, S., 2023. Capow: Context-aware ai-assisted proof of work based ddos defense. arXiv preprint .
- [16] Chandra, R., 2001. Parallel programming in OpenMP. Morgan kaufmann.
- [17] Chen, V., Das, S., Zhu, L., Malyar, J., McCann, P., 2015. Protocol to access white-space (paws) databases. Technical Report.
- [18] Chiriaco, V., Franzen, A., Thayil, R., 2017. Finding partial hash collisions by brute force parallel programming, in: 2017 IEEE Long Island Systems, Applications and Technology Conference (LISAT), IEEE.
- [19] Chor, B., Kushilevitz, E., Goldreich, O., 1998. Private information retrieval. Journal of the ACM (JACM) 45, 965–981.
- [20] Cooper, D., et al., 2024. Stateless hash-based digital signature standard .
- [21] Dang, T., Lichtinger, J., Liu, Y.K., Miller, C., Moody, D., Peralta, R., Perlner, R., 2024. Module-lattice-based digital signature standard .
- [22] Darzi, S., Yavuz, A.A., 2024a. 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.
- [23] Darzi, S., Yavuz, A.A., 2024b. Privacy-preserving and post-quantum counter denial of service framework for wireless networks, in: MIL-COM 2024- IEEE Military Communications Conference, IEEE.
- [24] Darzi, S., Yavuz, A.A., 2025. Slap: Secure location-proof and anonymous privacy-preserving spectrum access. preprint arXiv:2503.02019 .
- [25] Doriguzzi-Corin, R., Siracusa, D., 2024. Flad: adaptive federated learning for ddos attack detection. Computers & Security 137.
- [26] Federal Communications Commission, 2024. Citizens Broadband Radio Service. 47 CFR Part 96, Code of Federal Regulations.
- [27] Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Prest, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z., et al., 2018. Falcon: Fast-fourier lattice-based compact signatures over ntru. NIST's post-quantum cryptography standardization process 36.
- [28] Gao, Z., Zhu, H., Liu, Y., Li, M., Cao, Z., 2012. 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, pp. 1025–1027.
- [29] Geerling, J., . Power consumption benchmarks — Raspberry Pi Dramble. <https://www.pidramble.com/wiki/benchmarks/power-consumption>. Accessed: 2026-08-10.
- [30] Goldberg, I., 2007. Improving the robustness of private information retrieval, in: 2007 IEEE Symposium on Security and Privacy, IEEE.
- [31] Grissa, M., Yavuz, A.A., 2019. Location privacy in cognitive radios with multi-server private information retrieval. IEEE Trans. on Cognitive Comm. and Networking .
- [32] Grissa, M., Yavuz, A.A., Hamdaoui, B., 2016. Preserving the location privacy of secondary users in cooperative spectrum sensing. IEEE Transactions on Information Forensics and Security 12.
- [33] Grissa, M., Yavuz, A.A., Hamdaoui, B., Tirupathi, C., 2021. Anonymous dynamic spectrum access and sharing mechanisms for the cbrs band. IEEE Access 9, 33860–33879.
- [34] Günther, D., Heymann, M., Pinkas, B., Schneider, T., 2022. {GPU-accelerated}{PIR} with {Client-Independent} preprocessing for {Large-Scale} applications, in: 31st USENIX Security Symposium (USENIX Security 22), pp. 1759–1776.
- [35] Jakimoski, G., Subbalakshmi, K., 2008. Denial-of-service attacks on dynamic spectrum access networks, in: IEEE International Conference on Communications Workshops, IEEE. pp. 524–528.
- [36] Jasim, D.K., Sadkhan, S.B., 2021. 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. pp. 149–153.
- [37] Jing, L., Ke, L., Lei, Z., Xiaoya, Y., Yuanyuan, J., Huinan, J., 2025. Geohash coding location privacy protection scheme based on entropy weight topsis. The Journal of Supercomputing 81, 85.
- [38] Kelefouras, V., Kritikakou, A., Mporas, I., Kolonias, V., 2016. A high-performance matrix-matrix multiplication methodology for cpu and gpu architectures. The Journal of supercomputing 72.
- [39] Kirk, D., et al., 2007. Nvidia cuda software and gpu parallel computing architecture, in: ISMM, pp. 103–104.
- [40] Koo, J., Cha, H., 2010. Localizing wifi access points using signal strength. IEEE Communications letters 15, 187–189.
- [41] Lam, M., Johnson, J., Xiong, W., Maeng, K., Gupta, U., 2023. Gpu-based private information retrieval for on-device machine learning inference. arXiv preprint arXiv:2301.10904 .
- [42] Li, Q., Zong, R., 2025. Cat: A gpu-accelerated fhe framework with its application to high-precision private dataset query. arXiv preprint arXiv:2503.22227 .
- [43] Li, Y., Zhou, L., Zhu, H., Sun, L., 2015. Privacy-preserving location proof for securing large-scale database-driven cognitive radio networks. IEEE Internet of Things Journal 3, 563–571.
- [44] Nguyen-Thanh, N., Ta, D.T., Nguyen, V.T., 2019. Spoofing attack and surveillance game in geo-location database driven spectrum sharing. IET Communications 13, 74–84.
- [45] Patil, A., Iyer, S., López, O.L., Pandya, R.J., Pai, K., 2024. A comprehensive survey on spectrum sharing techniques for 5G/B5G intelligent wireless networks: Opportunities, challenges and future research directions. Computer Networks 253.
- [46] Rappaport, T.S., 2010. Wireless communications: Principles and practice, 2/E. Pearson Education India.
- [47] Raspberry Pi Ltd., . Raspberry pi 4 Model B specifications. <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/>. Accessed: 2026-08-10.
- [48] Robinson, M., Psaromiligkos, I., 2005. Received signal strength based location estimation of a wireless lan client, in: IEEE Wireless Communications and Networking Conference, IEEE.
- [49] Salami, Z., Ahmadian-Attari, M., Jannati, H., Aref, M.R., 2017. A location privacy-preserving method for spectrum sharing in database-driven cognitive radio networks. Wireless Personal Communications 95, 3687–3711.
- [50] of Standards, N.I., Technology, 2024. Fips 203 module-lattice-based key-encapsulation mechanism standard .
- [51] Szeplieniec, A., Ashur, T., Dhooche, S., 2020. Rescue-prime: a standard specification (sok). Cryptology ePrint Archive .
- [52] Tamrakar, S., Liu, J., Paverd, A., Ekberg, J.E., Pinkas, B., Asokan, N., 2017. The circle game: Scalable private membership test using trusted hardware, in: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, pp. 31–44.
- [53] Team, S., 2021. ethSTARK documentation-version 1.1. Technical Report. IACR preprint archive 2021.
- [54] The Tor Project, 2024. Tor metrics: Torperf. URL: <https://metrics.torproject.org/torperf.html>. accessed: 2024-01-01.
- [55] Troja, E., Bakiras, S., 2014. Leveraging p2p interactions for efficient location privacy in database-driven dynamic spectrum access, in: 22nd ACM SIGSPATIA Conf.
- [56] TU Darmstadt, 2024. Darmstadt svp challenge. <https://www.latticechallenge.org/svp-challenge/>. Accessed: 2024-04.
- [57] Ul Hassan, M., Rehmani, M.H., Rehan, M., Chen, J., 2022. Differential privacy in cognitive radio networks: a comprehensive survey. Cognitive Computation 14, 475–510.

- [58] Xin, J., Li, M., Luo, C., Li, P., 2016. Privacy-preserving spectrum query with location proofs in database-driven crns, in: 2016 IEEE Global Communications Conference (GLOBECOM), IEEE. pp. 1–6.
- [59] Xue, Y., Lu, X., Au, M.H., Zhang, C., 2024. Efficient linkable ring signatures: new framework and post-quantum instantiations, in: European Symposium on Research in Computer Security, Springer.
- [60] Zhang, L., Huang, J., Di, S., Matsuoka, S., Wahib, M., 2025. Can tensor cores benefit memory-bound kernels?(no!), in: Proceedings of the 17th Workshop on General Purpose Processing Using GPU, pp. 28–34.
- [61] Zhu, R., Xu, L., Zeng, Y., Yi, X., 2019. Lightweight privacy preservation for securing large-scale database-driven cognitive radio networks with location verification. *Security and Communication Networks* 2019, 9126376.