Received XX Month, XXXX; revised XX Month, XXXX; accepted XX Month, XXXX; Date of publication XX Month, XXXX; date of current version XX Month, XXXX.

Digital Object Identifier 10.1109/XXXX.2022.1234567

Bringing Private Reads to Hyperledger Fabric via Private Information Retrieval

First A. Author[1], Fellow, IEEE, Second B. Author[2], and Third C. Author Jr.[3], Member, IEEE

[1]National Institute of Standards and Technology, Boulder, CO 80305 USA
[2]Department of Physics, Colorado State University, Fort Collins, CO 80523 USA
[3]Electrical Engineering Department, University of Colorado, Boulder, CO 80309 USA

Corresponding author: First A. Author (email: author@ boulder.nist.gov).

This paragraph of the first footnote will contain support information, including sponsor and financial support acknowledgment. For example, "This work was supported in part by the U.S. Department of Commerce under Grant 123456."

ABSTRACT Permissioned blockchains ensure integrity and auditability of shared data but expose query parameters to peers during read operations, creating privacy risks for organizations querying sensitive records. This paper proposes a Private Information Retrieval (PIR) mechanism to enable *private reads* from Hyperledger Fabric's world state, allowing endorsing peers to process encrypted queries without learning which record is accessed. We implement and benchmark a PIR-enabled chaincode that performs ciphertext–plaintext ($ct{\times}pt$) homomorphic multiplication directly within *evaluate* transactions, preserving Fabric's endorsement and audit semantics. The prototype achieves an average end-to-end latency of 113 ms and a peer-side execution time below 42 ms, with approximately 2 MB of peer network traffic per private read in development mode—reducible by half under in-process deployment. Storage profiling across three channel configurations shows near-linear growth: block size increases from 77 KB to 294 KB and world-state from 112 KB to 332 KB as the ring dimension scales from 2^{13} to 2^{15}. Parameter analysis further indicates that ring size and record length jointly constrain packing capacity, supporting up to 512 records of 64 bytes each under 2^{15}. These results confirm the practicality of PIR-based private reads in permissioned ledgers and highlight future directions toward sublinear, constant-time, and verifiable query execution.

INDEX TERMS Private Information Retrieval (PIR), Hyperledger Fabric, Private Reads, Query Privacy.

I. INTRODUCTION

A. MOTIVATION AND PROBLEM STATEMENT

Blockchain [1] is a peer-to-peer network protocols that uses cryptographic primitives and consensus mechanism to create and maintain a distributed ledger of transactions, such as Ethereum [2] and Hyperledger Fabric [3].

Hyperledger Fabric (HLF or Fabric) is widely adopted in various domains, one of them being Cyber Threat Intelligence (CTI) sharing [4]–[7], where multiple organizations exchange threat indicators and wish to keep their queries confidential not to be assosiated with specific threats or incidents. Blockchain guarantees, however, primarily cover data integrity and auditability, not query privacy. This issue was recently highlighted by the Ethereum's privacy special interest group in their "PSE Roadmap: 2025 and Beyond" [8], where they also emphasized the need for *private reads* next to private writes and private voting.

In Hyperledger Fabric [3], the separation of *evaluate* and *submit* makes the read-privacy gap explicit: an *evaluate* call is a read-only proposal sent to endorsing peers, which execute the chaincode and return results without committing to the ledger [9]. Crucially, these peers still observe all function arguments and read-sets. Thus, in multi-organization settings, the dominant privacy risk arises not from the immutable ledger, but from endorsing peers who can log or infer sensitive query information.

This motivates us to target *read* privacy issue in Fabric, defined as the ability to hide which record is being queried from endorsing peers during *evaluate* type calls. Given problem description, an obvious approach is to construct and use Private Information Retrieval (PIR) [10] protocol, which would enable client to retrieve an item from a world state database without revealing to peer which item was requested. However, integrating PIR directly into Fabric's chaincode layer introduces practical challenges related to performance, communication overhead, and parameter tuning.

B. CHOOSING AN APPROPRIATE PIR SCHEME

PIR protocols can broadly be categorized into two families: *Information-Theoretic* (IT-PIR) and *Computational* (CPIR) schemes. Later, each familly can be further divided into: *non-preprocessing* schemes that perform all computation online, and *preprocessing-based* schemes that split the protocol into offline and online phases.

Information-Theoretic. IT-PIR schemes [10]–[14] provide unconditional privacy by distributing the database across **multiple non-colluding servers**, a client then queries subsets of servers such that no single server learns the selection index. Though IT-PIR schemes offer strong privacy guarantees and low communication overhead, they require multiple non-colluding servers, which is often impractical in consortium blockchains where all peers are typically honest-but-curious and collusion is a realistic threat.

PIR without Preprocessing. Older [15]–[19] and more recent non-preprocessing single-server schemes such as SealPIR [20], FastPIR [21], OnionPIR [22], and Spiral [23] rely on cryptographic assumptions, such as homomorphic encryption (HE) to achieve privacy using **single untrusted server**. These constructions require only one round of interaction—an encrypted query sent to the server and an encrypted response returned—making them well-suited to Fabric's *evaluate* model, where minimal state is preserved between calls and each peer executes chaincode statelessly. While HE-based protocols are computationally heavier, they avoid persistent client-specific state and additional communication rounds, which are incompatible with Fabric's endorsement flow and deterministic transaction semantics.

PIR with Preprocessing. Beimel et al. [24] introduced the concept of PIR with preprocessing, where a preprocessing (offline) phase is performed before the actual query (online) phase to reduce online communication and computation costs. Recent advances such as SimplePIR [25], Piano [26], HintlessPIR [27], and YPIR [28] achieve sublinear or near-constant online time by introducing an offline phase where the client or server precomputes query-independent data, usually called "hints". Other doubly-efficient constructions [29]–[31] shift most computation to preprocessing, trading online efficiency for large offline communication or storage. While appealing for cloud-hosted servers, these techniques are ill-suited for permissioned blockchains for two reasons: (i) if the hints are stored client-side, each participant must predownload large portions of the world state to generate them, defeating Fabric's lightweight-client model; and (ii) if hints are stored peer-side, the chaincode or world state would need to maintain per-client data, violating Fabric's stateless transaction design. Moreover, these schemes introduce significant storage blowup: for instance, SimplePIR's [25] precomputation tables or HintlessPIR's [27] server hints can exceed several times the original database size, and Piano's [26] client preprocessing requires full-database downloads in its initialization stage. Such assumptions are incompatible with blockchain environments, where deterministic, replayable, and stateless transaction execution is critical.

Although less computationally efficient in theory and limited in database capacity due to polynomial packing constraints, we argue that HE-based PIR is the most viable option for enabling *private reads* in Fabric-like permissioned ledgers, for the following reasons: (i) it requires no changes to Fabric's consensus, core architecture or endorsement policies, (ii) it avoids the need for multiple non-colluding servers or endorsing peers in our case, (iii) it maintains Fabric's stateless transaction model without per-client state on peers, (iv) it avoids offline preprocessing phases incompatible with Fabric's on-demand *evaluate* calls, (v) it avoids offline communication or storage blowup on clients or peers, (vi) it supports single-round query-response interactions naturally compatible with Fabric's *evaluate* calls, (vii) it leverages existing homomorphic encryption libraries that can be integrated into chaincode, and (viii) it achieves somewhat practical performance under certain parameter choices, as we demonstrate in this work.

We further argue that the limitations of HE-based PIR, particularly its database capacity constraints due to polynomial packing, are an acceptable trade-off for enabling *private reads* in permissioned ledgers handling smaller amounts of more sensitive records in multi-organization settings.

C. RELATED WORK

We now review prior efforts to integrate PIR with blockchain and distributed systems to enhance query privacy. Detailed comparison Table 10 is available in Section V.

Kumar et al. introduced a series of HLF frameworks that integrate Private Information Retrieval (PIR) across different domains, namely BRON [32] and DEBPIR [33]. Although posed as enabling PIR functionality, their evaluations primarily focus on the system's write throughput for general transactions rather than the specific performance of PIR queries.

Xiao et al. [34] propose Cloak, a privacy-preserving blockchain query scheme that uses distributed point functions (DPF) and noise-based sub-requests to hide retrieval information, achieving communication costs of 0.2-0.5 KB and query latency of 0.4-0.5 ms for databases with 4-64 records of 16-32 bytes each.

Mazmudar et al. [35] propose Peer2PIR, a privacy solution for IPFS using a combination of PAILLIERPIR and RLWEPIR schemes to hide query content across peer routing, provider advertisements, and content retrieval functionalities. For relevant private content retrieval function, they achieve communication costs of 10-15 MB and latencies of 0.5-15 s for databases of 1,000 to 1,000,000 records with 256 KB content blocks using the Spiral [23] protocol.

Kaihua et al. [36] design an SPV protocol for lightweight Bitcoin clients using a hybrid PIR scheme (combining IT-PIR and C-PIR) to enhance query privacy over Bloom-filter-based methods. Their system uses temporally-partitioned databases (Address, Merkle Tree, and Transaction DBs) to optimize performance. For a single transaction verification in the weekly database, it achieves a communication cost of 666 KB and latency of 2.84 s for databases with 7,688-512,460 records of 62-876 bytes each.

Targeted gap. Blockchains like Hyperledger Fabric ensure integrity and auditability of shared data but leak function arguments and read-sets to endorsing peers during both *evaluate* and *submit* calls, creating privacy risks in multi-organisation settings. While prior works have explored PIR integration in blockchain contexts, none have specifically addressed the challenge of enabling *private reads* directly within Fabric's chaincode layer. To fill this gap, we explore how to enable *private reads* from Fabric's world state by integrating a HE-based PIR mechanism directly into chaincode, and we evaluate its practicality through detailed benchmarks and parameter studies.

D. KEY OBJECTIVE AND CONTRIBUTIONS

In this work, we demonstrate that HE-based PIR can be implemented natively in chaincode to enable *private reads* in Hyperledger Fabric under *evaluate* calls and achieve somewhat practical performance under certain parametrization choices.

Hence, the main contributions of this work are:

1) **First PIR-on-Blockchain system for private reads.** We present the first end-to-end design, implementation, and evaluation of a HE-based PIR mechanism integrated directly into Hyperledger Fabric chaincode, enabling private reads from world state without modifying Fabric's core architecture.
2) **Polynomial database construction.** We derive necessary conditions on the cryptographic parameters, database size, and record structure to encode domain-specific structured key-value records into plaintext-polynomial format suitable for HE-based PIR, ensuring retrieval correctness and cryptographic feasibility.
3) **Multi-channel approach.** To overcome database size limits imposed by polynomial packing constraints, we propose a multi-channel architecture where each channel hosts a distinct parameter set and a record template, allowing clients to select the appropriate channel based on their query needs.
4) **Comprehensive evaluation.** We implement a prototype using the Lattigo [37] library and benchmark cryptographic operations, blockchain interactions, and end-to-end query latency under various configurations. Our results show that single-query latencies remain practical for typical Fabric deployments.
5) **Open-Source Release.** To encourage reproducibility and use by other researchers, we open-source the complete CPIR-on-Blockchain system, including chaincode, client, and experimental setup. The repository is available at: https://github.com/artias13/2_2_HLF_CPIR.

E. ORGANIZATION

The remainder of this paper is organized as follows: Section II reviews Fabric privacy features, database limits, Homomorphic Encryption-based PIR, and notation. Section III describes our system and threat models, polynomial database construction, feasibility constraints, multi-channel architecture, workflow and algorithms. Section IV shows experimental setup, cryptographic and blockchain benchmarks, and overall system performance. Section V discusses limitations and future directions, and Section VI concludes the paper.

II. PRELIMINARIES

A. FABRIC NATIVE PRIVACY FEATURES

We hereby acknowledge several native solutions for privacy that Hyperledger Fabric provides and explain why they do not fully address the query privacy problem.

Separate Channels. Fabric's multi-channel architecture [9] isolates ledgers across subgroups of organizations, limiting which participants observe which data. However, channel separation controls *who* sees a ledger, query intent remains visible to all endorsers of a channel.

Private Data Collections (PDC). PDCs [9] restrict which organizations store and access private key–value pairs. The shared ledger records only hashes, while members of the collection hold plaintext. PDCs provide access control but still expose function arguments to endorsers inside the collection, leaving query patterns observable.

Fabric Private Chaincode (FPC). FPC [9], [38] executes chaincode within Intel SGX enclaves. Arguments and state are protected even from peer operators, but this requires Trusted Execution Environments (TEEs) and attestation, introducing additional hardware and trust assumptions.

B. FABRIC STORAGE LIMITS

We here address the question of how large a database can be stored in Fabric world state and ledger history, as this impacts the practical limits of our PIR construction.

World state (LevelDB/CouchDB). Fabric imposes no hard limit on the number of key–value entries in world state; capacity depends on available disk space and peer I/O throughput. In our implementation, each channel maintains a few small artifacts in world state that amount up to a 350 KB under typical parameters (see Section IV), LevelDB is sufficient and preferable for performance and simplicity, while CouchDB [9] remains an option to extend from 8 MB to 4 GB if needed.

Ledger history (blockchain log). According to the documentation [9], block size is constrained by the ordering service configuration. By default, the Fabric orderer limits the serialized payload to *AbsoluteMaxBytes = 10 MB* (recommended under 49 MB given the gRPC ceiling of 100 MB), and typically aggregates up to *MaxMessageCount = 500* transactions per block or *PreferredMaxBytes = 2 MB*. In our system, these limits affect only *submit* transactions such as *InitLedger* or record updates. *Evaluate* transactions (including *PIRQuery*) are read-only and do not generate blocks, thus unaffected by ordering or batching constraints.

Implication. The effective capacity of a channel is governed primarily by cryptographic feasibility (Fig. 3) and the size of a single world-state value (i.e., m_{DB}), rather than by Fabric's block or database limits. For large or binary CTI objects (e.g., malware samples or full PCAPs), an optional extension is to store them off-chain in IPFS [39] while persisting only their content identifiers (CIDs) in world state, keeping the polynomial m_{DB} as the structured component used for private retrieval.

C. PRIVATE INFORMATION RETRIEVAL (PIR)

Homomorphic Encryption. Lattice-based homomorphic encryption schemes, such as Brakerski/ Fan-Vercauteren (BFV), Brakerski-Gentry-Vaikuntanathan (BGV), CKKS and TFHE [40]–[43] have emerged as foundational technologies for practical CPIR implementations.

We base our CPIR construction on the BGV scheme, a lattice-based homomorphic encryption based on the Ring Learning With Errors (RLWE) problem [44], which supports both addition and multiplication over ciphertexts. The BGV scheme defines operations over two polynomial rings: a ciphertext ring $R_Q = \mathbb{Z}_Q[X]/(X^N + 1)$ and a plaintext ring $R_T = \mathbb{Z}_T[X]/(X^N + 1)$, both sharing the same dimension $N = 2^{\log N}$. In our implementation, these rings are jointly specified by a single parameter literal $(\log N, \log Q_i, \log P_i, T)$ as provided by the Lattigo library [37] and paper [45]. The field T determines R_T, while the modulus chain (Q, P) and their bit-lengths $(\log Q_i, \log P_i)$ determine R_Q.

CPIR Definition. Model the database as a vector $D = \{d_0, \ldots, d_{n-1}\}$. To retrieve desired record d_i without revealing i, the client forms a one-hot selection vector $\hat{v}_i$ defined as $\hat{v}_i = (0, \ldots, 0, 1, 0, \ldots, 0)$ where 1 corresponds to the desired record at index i. He then encrypts it as $ct_q = \mathrm{Enc}_{pk}(\hat{v}_i)$ under a public key pk and sends ct_q to the server, which computes $ct_r = (ct_q \cdot D) = \mathrm{Enc}_{pk}(d_i)$, returning ct_r to the client for decryption $d_i = \mathrm{Dec}_{sk}(ct_r)$ using his secret key sk, thus obtaining the desired record d_i without revealing i to the server, or peer in our case.

CPIR Instantiation: We instantiate Computational PIR as a tuple of probabilistic polynomial-time algorithms (*KeyGen*, *Enc*, *Eval*, *Dec*):

- $\mathit{KeyGen}(\lambda) \to (pk, sk)$: On input the security parameter λ, output a public key pk and a secret key sk.
- $\mathit{Enc}_{pk}(\hat{v}_i) \to ct_q$: Given a windowed selection vector $\hat{v}_i \in \{0,1\}^N$, encode it into the plaintext ring R_T and encrypt to a query ciphertext ct_q under pk.
- $\mathit{Eval}(ct_q, m_{DB}) \to ct_r$: Given ct_q and the plaintext polynomial database $m_{DB} \in R_T$, homomorphically evaluate the product to obtain an encrypted response $ct_r \in R_Q$.
- $\mathit{Dec}_{sk}(ct_r) \to d_i$: Using the secret key sk, decrypt the response ciphertext ct_r to recover the desired record d_i.

Protocol objective. Correctness requires that for all $i \in [n]$,

$$\mathit{Dec}_{sk}(\mathit{Eval}(\mathit{Enc}_{pk}(\hat{v}_i), m_{DB})) = d_i.$$

Remark (restricted operation set). The full BGV scheme also provides EvalKeyGen to generate relinearization and rotation keys, supporting ciphertext–ciphertext multiplication, automorphisms, and modulus switching. Our PIR construction requires only ciphertext–plaintext multiplication ($ct \times pt$), since the database polynomial m_{DB} is kept in plaintext within world state. This avoids degree growth and level management, so we do not expose EvalKeyGen in chaincode. Extending to ciphertext–ciphertext evaluation would require additional on-chain artifacts (relinearization keys, Galois keys, encrypted m_{DB}), larger world-state footprint, and higher evaluation cost, which we leave as future work.

D. NOTATION

We summarize the main notation used throughout the paper in Table 1.

III. PROPOSED SYSTEM

A. SYSTEM MODEL

We introduce a blockchain-based query privacy system designed for permissioned ledgers. The system enables clients to perform *private reads* from the ledger while endorsing peers can evaluate read-only queries over encrypted inputs without learning which record was accessed. We achieve this by integrating a lattice-based CPIR scheme based on the BGV [41] homomorphic encryption scheme directly into Fabric chaincode. This approach ensures that clients remain the sole holders of decryption keys, while peers perform only black-box computations, thereby enhancing overall privacy without requiring trusted hardware or protocol modifications. Our system is composed of the following entities:

- **Data Owner ($\mathcal{DO}$):** Endorsing peers that hold the current plaintext polynomial m_{DB} in world state and execute PIR during *evaluate*. $\mathcal{DO}$ is honest-but-curious.
- **Data Writer ($\mathcal{DW}$):** A client organization that provisions or refreshes the database. $\mathcal{DW}$ invokes *submit* to initialize the ledger (e.g., set n and template bounds). Chaincode computes $record_s$, packs $D = \{d_0, \ldots, d_{n-1}\}$, encodes it into m_{DB}, and persists it.
- **Data Requester ($\mathcal{DR}$):** A client that privately retrieves a record. $\mathcal{DR}$ runs $\mathit{KeyGen}(\lambda) \to (pk, sk)$, forms $ct_q = \mathit{Enc}_{pk}(v_i)$, calls *evaluate PIRQuery*, and later decrypts ct_r.
- **Gateway ($\mathcal{GW}$):** The Fabric client/chaincode interface used by $\mathcal{DW}$ and $\mathcal{DR}$ to invoke *InitLedger*, *Get-*

TABLE 1. Notation

Symbol	Description
λ	Security parameter
n	Database size; index domain $[n] = \{0, \dots, n-1\}$
D	Database records $\{d_0, \dots, d_{n-1}\}$
m_{DB}	Plaintext polynomial representation of D
$\hat{v}_i$	One-hot selector for index i
v_i	Windowed selector for index i with $record_s$ contiguous ones
$Enc_{pk}(\cdot)$, $Dec_{sk}(\cdot)$	Encrypt / Decrypt
$Eval(\cdot)$	Homomorphic evaluation (ct–pt multiply)
pk, sk	Public / secret keys
ct_q	Encrypted query $Enc_{pk}(\hat{v}_i)$
ct_r	Encrypted response $Eval(ct_q, m_{DB})$
d_i	Decrypted record $Dec_{sk}(ct_r)$
$KeyGen(\lambda)$	Key generation $\rightarrow (pk, sk)$
N	Ring dimension
$\log N$	Logarithm base 2 of ring dimension N
$\log Q_i$	Bit-lengths of primes forming modulus chain Q
$\log P_i$	Bit-lengths of special primes P
T	Plaintext modulus
$record_s$	Slots allocated per record
$record_b$	Base serialized size of a record in bytes
$record_{\mu,\log N}$	Template-specific minimum
$\mathcal{S}$	Allowed discrete slot sizes
c	Coefficient vector $(c_0, \dots, c_{N-1})$
$\lvert \cdot \rvert$, size$(\cdot)$	Length in elements / size in bytes
$\mathsf{Cap}(N, s, n)$	Capacity predicate: $n \cdot s \le N$
$\mathsf{Min}(\log N, s)$	Template predicate: $s \ge record_{\mu,\log N}$
$\mathsf{Disc}(s)$	Discrete predicate: $s \in \mathcal{S}$
$\mathsf{Cap} \wedge \mathsf{Min} \wedge \mathsf{Disc}$	Feasibility condition
$\mathcal{DO}$, $\mathcal{DW}$, $\mathcal{DR}$, $\mathcal{GW}$	Data Owner; Data Writer; Data Requester; Gateway
evaluate, *submit*	Fabric read / write transaction types
$\mathcal{L}$	Leakage considered (ciphertext size, protocol timing)

Metadata, and *PIRQuery*. It follows standard Fabric semantics; no extra trust is assumed.

Remark (world-state scope). In Fabric, the "ledger" comprises the blockchain log and the world state. Our CPIR operates on the world state: m_{DB} encodes the latest key–value snapshot, not the historical transaction logs. Both *world state* and *ledger* capacity concerns are addressed in Section II.

B. THREAT MODEL

Our design follows the standard *honest-but-curious* adversarial model. We explicitly consider the following assumptions and threats:

- **Endorsing peers ($\mathcal{DO}$).** Execute chaincode correctly but may try to infer the queried index from *evaluate* inputs or logs. They see ct_q, metadata, and m_{DB}.
- **Data Writer ($\mathcal{DW}$).** Issues initialization writes via *submit*. $\mathcal{DW}$ is not trusted with decryption keys and learns nothing about $\mathcal{DR}$'s queries. We assume $\mathcal{DW}$ follows the write protocol but is not relied upon for privacy.
- **External observers.** May eavesdrop on client–peer traffic. Without sk, ct_q and ct_r reveal nothing under BGV assumptions.

Security objective. For any $i \in [n]$, neither $\mathcal{DO}$ nor external observers can distinguish which d_i is requested from ct_q and ct_r. The only permissible leakage is ciphertext size and protocol timing, denoted collectively as $\mathcal{L}$.

C. SYSTEM OVERVIEW

The proposed system integrates computational Private Information Retrieval (CPIR) directly into Hyperledger Fabric chaincode. Its purpose is to ensure that query indices remain hidden from endorsing peers while preserving Fabric's endorsement and audit workflow. At a high level, the workflow consists of four stages, illustrated in Fig. 1.

1) **Ledger initialization.** $\mathcal{DW}$ invokes *InitLedger* via $\mathcal{GW}$ using *submit*. Chaincode derives $record_s$ from $record_b$, packs D into $c = (c_0, \dots, c_{N-1})$, encodes m_{DB}, and stores m_{DB} and metadata in world state held by $\mathcal{DO}$.
2) **Metadata discovery.** $\mathcal{DR}$ calls *GetMetadata* via *evaluate* to obtain n, $record_s$, and BGV parameters needed to form a valid query.
3) **Private retrieval.** $\mathcal{DR}$ constructs $ct_q = Enc_{pk}(v_i)$ and invokes *PIRQuery* via *evaluate*. $\mathcal{DO}$ computes $ct_r = Eval(ct_q, m_{DB})$ and returns it.
4) **Decryption.** $\mathcal{DR}$ decrypts ct_r to recover $d_i = Dec_{sk}(ct_r)$.

D. POLYNOMIAL DATABASE CONSTRUCTION

To enable PIR queries over structured ledger data, we must embed records into a plaintext polynomial m_{DB} suitable for BGV evaluation. Our prototype adopts a fixed-width packing strategy, illustrated in Fig. 2, which proceeds in four steps.

Step 1: Serialize to Byte Array. Each ledger record d_i is serialized as a UTF-8 byte array. In our motivating use case of Cyber Threat Intelligence (CTI) sharing, JSON objects containing fields such as hash digests and threat

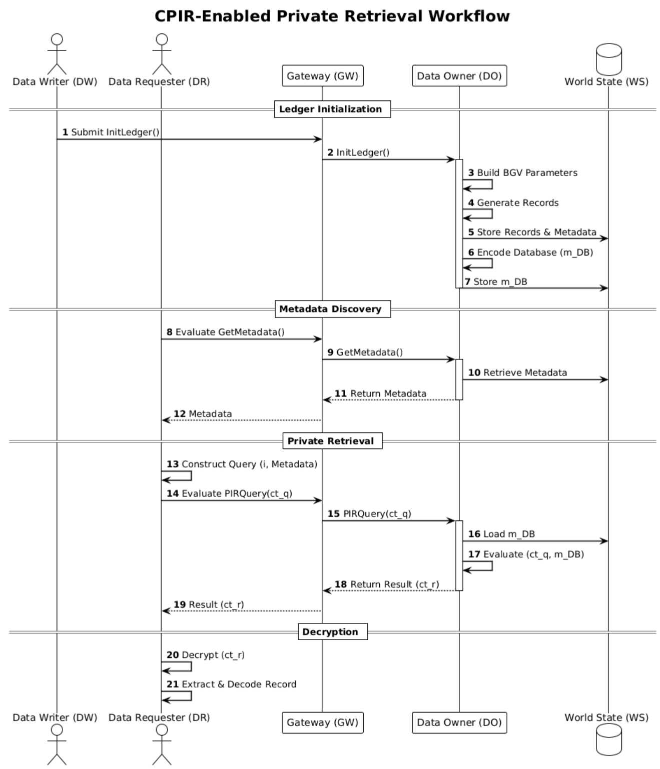

FIGURE 1. **"Workflow. $\mathcal{DW}$ initializes the ledger via $\mathcal{GW}$, which triggers chaincode on endorsing peers ($\mathcal{DO}$). $\mathcal{DO}$ executes the protocol and persists state in world state (m_{DB}, metadata, JSON records). $\mathcal{DR}$ later obtains metadata, submits $ct_q = Enc_{pk}(v_i)$, $\mathcal{DO}$ evaluates $ct_r = Eval(ct_q, m\mathrm{DB})$ against world state, and $\mathcal{DR}$ decrypts to d_i.**

levels are flattened into byte sequences. Every character is represented by its ASCII code in $[0, 255]$. This ensures that arbitrary structured records can be embedded in the polynomial without loss of information.

Step 2: Calculate Slot Window. We determine the slot allocation per record as:

$$record_s = \left\lceil \frac{record_b}{bytesPerSlot} \right\rceil,$$

where $record_b$ is the maximum serialized record length observed in bytes. In our prototype, each slot stores exactly one byte. For example, if the largest record is 126 bytes, then $record_s = \lceil 126/1 \rceil = 126$, which rounds up to 128 slots due to the discrete window policy. This guarantees uniform slot windows across all records, simplifying query construction at the cost of potential padding overhead.

Step 3: Pack into Coefficient Vector. Serialized byte arrays are inserted into disjoint slot windows of length $record_s$ within a coefficient vector $c = (c_0, c_1, \ldots, c_{N-1})$, where $N = 2^{\log N}$ is the ring capacity of the BGV scheme. Padding zeros are added if a record is shorter than $record_s$. Thus each record d_i occupies a contiguous slot interval that can be privately retrieved through PIR.

Step 4: Encode into Polynomial. Finally, the coefficient vector d is encoded into a plaintext polynomial:

$$m_{DB}(X) = \sum_{j=0}^{N-1} c_j X^j \in R_t,$$

where $R_t = \mathbb{Z}_t[X]/(X^N + 1)$. This polynomial serves as the plaintext database representation stored in the Fabric world state. Endorsing peers operate over m_{DB} during PIR queries, while clients recover only the slots corresponding to their requested record.

E. FEASIBILITY CONSTRAINTS

Embedding records into the plaintext polynomial m_{DB} is feasible only for parameter triples $(\log N, n, record_s)$ that satisfy *all* of the following constraints. These constraints form a hierarchical relationship: template requirements dominate discrete allocation, and both are ultimately bounded by ring capacity.

Constraint 1: Ring capacity. The total number of occupied slots cannot exceed the ring size:

$$n \cdot record_s \ \leq \ N.$$

This represents the fundamental mathematical limit imposed by the cryptographic parameters. For example, with $\log N = 13$ ($N = 8192$) and $record_s = 224$, at most $\lfloor 8192/224 \rfloor = 36$ records can be packed.

Constraint 2: Template-specific minima. Each security level ($\log N$) corresponds to a record template with mandatory fields that impose a minimum slot requirement $record_{\mu,\log N}$. Examples include:

- **Mini records** ($\log N = 13$): MD5 hash + malware family + threat level $\Rightarrow$ $record_{\mu,13} \approx 128$ bytes.
- **Mid records** ($\log N = 14$): MD5 + SHA-256 short + malware class + AV detects $\Rightarrow$ $record_{\mu,14} \approx 224$ bytes.
- **Rich records** ($\log N = 15$): MD5 + full SHA-256 + all metadata $\Rightarrow$ $record_{\mu,15} \approx 256$ bytes.

These minima arise from generator checks of the form:

$$\begin{aligned} \text{Mini:} &\quad record_s \ \geq \ record_b + 32 + 15, \\ \text{Mid:} &\quad record_s \ \geq \ record_b + 32 + 16 + 15, \\ \text{Rich:} &\quad record_s \ \geq \ record_b + 32 + 64 + 15, \end{aligned}$$

where $record_b \approx$ 113–125 bytes denotes the base JSON structure and numeric terms represent mandatory hash fields and serialization overhead.

Constraint 3: Discrete allocation policy. Operationally, we restrict the slot window $record_s$ to a discrete set for implementation simplicity:

$$\mathcal{S} = \{64, 128, 224, 256, 384, 512\} \text{ bytes.}$$

This means that even if Constraints 1 and 2 are satisfied, the configuration is rejected unless $record_s \in \mathcal{S}$.

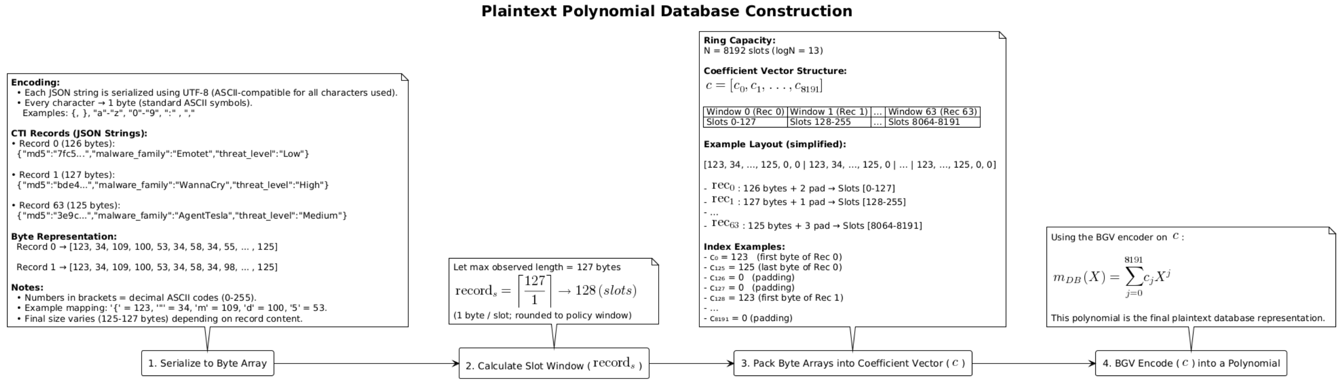

FIGURE 2. m_{DB} **construction from JSON to plaintext polynomial. Each record is serialized to bytes, mapped into a fixed slot window** $record_s$**, and packed into a coefficient vector** c**. The vector is then encoded as a BGV plaintext polynomial** m_{DB}**, which is stored in the Fabric world state.**

Constraint hierarchy and feasibility. Feasibility of a configuration $(\log N, n, record_s)$ is defined by three predicates:

$$\begin{aligned}\mathsf{Cap}(N,s,n) &: n \cdot s \le N \quad \text{(ring capacity)}\\ \mathsf{Min}(\log N,s) &: s \ge record_{\mu,\log N} \quad \text{(template minimum)}\\ \mathsf{Disc}(s) &: s \in \mathcal{S} \quad \text{(discrete allocation).}\end{aligned}$$

where $s = record_s$ and $N = 2^{\log N}$. A configuration is feasible iff:

$$\mathsf{Cap}(N,s,n) \ \wedge\ \mathsf{Min}(\log N,s) \ \wedge\ \mathsf{Disc}(s).$$

Examples. The interaction of these predicates is illustrated in Fig. 3. We provide three concrete examples below:

a) Polynomial degree $\log N = 13$ (Mini):

- $\mathsf{Min}(13,s)$ enforces $s \ge record_{\mu,13} \approx 128 \Rightarrow$ the smallest candidate is $s = 128$.
- $\mathsf{Disc}(s)$ requires $s \in \mathcal{S} \Rightarrow 128$ is allowed (while 64 is invalid).
- $\mathsf{Cap}(8192,128,n)$ gives $n \le \lfloor 8192/128 \rfloor = 64$.
- *Feasible:* $(\log N = 13, s = 128, n \le 64)$.

b) Polynomial degree $\log N = 14$ (Mid):

- $\mathsf{Min}(14,s)$ enforces $s \ge record_{\mu,14} \approx 224 \Rightarrow$ smallest candidate is 224.
- $\mathsf{Disc}(s)$ allows $224, 256, \ldots$ but excludes smaller windows.
- $\mathsf{Cap}(16384,224,n)$ gives $n \le \lfloor 16384/224 \rfloor = 73$.
- *Feasible:* $(\log N = 14, s \in \{224, 256, \ldots\}, n \le 73)$.

c) Polynomial degree $\log N = 15$ (Rich):

- $\mathsf{Min}(15,s)$ enforces $s \ge record_{\mu,15} \approx 256$.
- $\mathsf{Disc}(s)$ excludes $64, 128, 224$; smallest valid is 256.
- $\mathsf{Cap}(32768,256,n)$ gives $n \le \lfloor 32768/256 \rfloor = 128$.
- *Feasible:* $(\log N = 15, s \ge 256, n \le 128)$.

Implications. Increasing $\log N$ raises ring capacity N and thus n, but also requires larger $record_s$ if given richer templates. Feasible configurations occur only where all 3 predicates are met, guiding practical parameter selection.

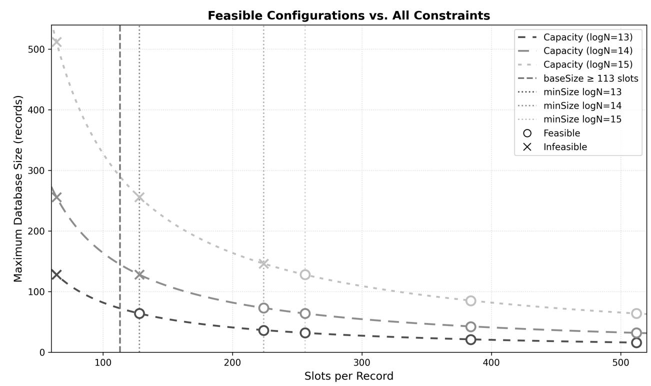

FIGURE 3. **Feasible configurations under the joint constraints** Cap, Min, **and** Disc**. Dashed curves show ring-capacity limits for** $\log N \in \{13, 14, 15\}$**, vertical lines mark template-driven minima** $record_{\mu,\log N}$**, and x-axis ticks correspond to discrete slot sizes** $\mathcal{S}$**. Circles indicate feasible triples** $(\log N, record_s, n)$**.**

F. MULTI-CHANNEL ARCHITECTURE

The packing strategy and feasibility constraints highlight an important observation: no single homomorphic parameter set can efficiently support the full diversity of Cyber Threat Intelligence (CTI) record formats. Compact records fit comfortably under smaller rings, while full JSON objects with long cryptographic hashes exceed the slot budget of these configurations. To balance scalability and expressiveness, we design a *multi-channel architecture* in Hyperledger Fabric (Fig. 4), where each channel is provisioned with a distinct BGV parameter set and record template.

a) Channel Mini ($\log N = 13$). Supports compact CTI records (e.g., MD5, malware family, threat level) with maximum scalability and lowest query latency. For example, with $N = 8192$ slots, the system accommodates up to 128 records when $\max_i |d_i| \le 64$ bytes, and 16 records when $\max_i |d_i| \le 512$ bytes.

b) Channel Mid ($\log N = 14$). Targets medium-sized records that include MD5 and truncated SHA-256 fields

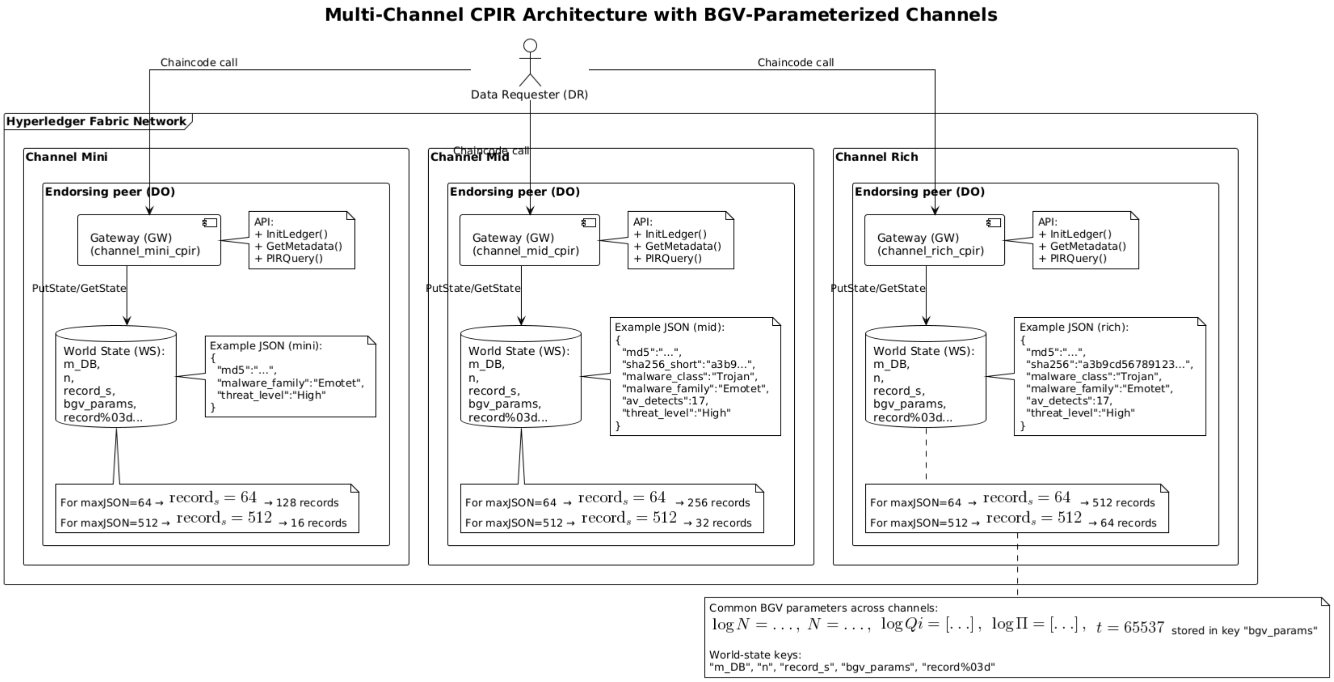

FIGURE 4. Multi-channel CPIR architecture. Each channel instantiates a separate CPIR chaincode and maintains its own m_{DB} polynomial, parameterized by $\log N$. This allows compact, mid-size, and rich CTI records to coexist under the same Fabric network.

alongside classification metadata. With $N = 16384$ slots, the system supports up to 256 records at ≤ 64 bytes or 32 records at ≤ 512 bytes.

c) Channel Rich ($\log N = 15$). Handles the most detailed records, including full-length hashes and multiple metadata fields. Here, $N = 32768$ slots allow up to 512 records at ≤ 64 bytes or 64 records at ≤ 512 bytes.

Channel semantics. As shown in Fig. 4, each channel maintains its own PIR chaincode instance and world state. The world state contains:

- The *polynomial view*: the packed plaintext polynomial m_{DB} under key *"m_DB"*.
- The *normal view*: JSON records stored optionally under keys *"record%03d"* for auditability.
- *Metadata*:
  - *"n"*: number of records n,
  - *"record_s"*: slots per record $record_s$,
  - *"bgv_params"*: $\{\log N, N, \log Q_i, \log P_i, T\}$.

G. WORKFLOW DETAILS

The complete workflow of our CPIR-enabled Fabric system is driven by five algorithms (Alg. 1–5), corresponding to the chaincode interface (*InitLedger*, *GetMetadata*, *PIRQuery*) and the client-side routines (*FormSelectionVector*, *DecryptResult*). Figure 1 provides the high-level overview. A Data Writer ($\mathcal{DW}$) provisions the database $D = \{d_0, \ldots, d_{n-1}\}$, a Data Owner ($\mathcal{DO}$) maintains the polynomial m_{DB} in world state and executes PIR evaluations, and a Data Requester ($\mathcal{DR}$) retrieves d_i privately using homomorphic encryption. The detailed steps are as follows:

1) **DW submits initialization.** $\mathcal{DW}$ calls *InitLedger* (Alg. 1) via $\mathcal{GW}$ (*submit*) with inputs $(n, record_s^{\mathcal{DW}})$ and an optional hint $(\log N, \log Q_i, \log P_i, T)$. Here $record_s^{\mathcal{DW}}$ denotes the maximum JSON size anticipated by the writer.
2) **DO validates and derives parameters.** $\mathcal{DO}$ rounds the writer's input to a discrete slot size $record_s^{\mathcal{GW}} = 8 \cdot \lceil record_s^{\mathcal{DW}}/8 \rceil$. If $\log N$ is absent, the smallest feasible $\log N$ is chosen such that $\mathsf{Cap}(N, record_s^{\mathcal{GW}}, n)$ holds. BGV parameters $\{\log N, N, \log Q_i, \log P_i, T\}$ are constructed and stored.
3) **DO prepares records.** Records $D = \{d_0, \ldots, d_{n-1}\}$ are ingested or synthesized with $|d_i| \leq record_s^{\mathcal{DW}}$. The definitive slot allocation is then fixed as $record_s = 8 \cdot \lceil \max_i |d_i|/8 \rceil$ (discrete policy), checked against feasibility predicates $\mathsf{Min}(\log N, record_s)$, $\mathsf{Disc}(record_s)$, $\mathsf{Cap}(N, record_s, n)$.
4) **Pack and persist.** Each d_i is placed in a disjoint window $J_i = \{i \cdot record_s, \ldots, (i+1)record_s - 1\}$ of $c = (c_0, \ldots, c_{N-1})$, zeros pad unused slots, and the polynomial $m_{DB}(X) = \sum c_j X^j$ is encoded at max level. World state stores *"m_DB"*, *"n"*, *"record_s"*, and *"bgv_params"* plus optional *"record%03d"* entries.
5) **DR discovers metadata.** $\mathcal{DR}$ calls *GetMetadata* (Alg. 2) via *evaluate* to obtain

$(n, record_s, \log N, N, T, \log Q_i, \log P_i)$. This enables reconstruction of the cryptographic context.

6) **DR instantiates crypto context.** From metadata, $\mathcal{DR}$ builds parameters, executes *KeyGen*$(\lambda) \rightarrow (pk, sk)$, and prepares encoder/encryptor objects.
7) **Form and encrypt query.** For index $i \in [n]$, $\mathcal{DR}$ runs *FormSelectionVector* (Alg. 3): define J_i, set v_i with ones on J_i, encode to $m_q(X)$, encrypt as $ct_q = Enc_{pk}(v_i)$, and serialize/Base64-encode.
8) **PIR query evaluation.** $\mathcal{DR}$ issues *PIRQuery*(ct_q^{B64}) (Alg. 4) via *evaluate*. $\mathcal{DO}$ decodes, reloads m_{DB} if necessary, and computes $ct_r = Eval(ct_q, m_{DB})$, returning the Base64-encoded ciphertext.
9) **Decryption and reconstruction.** $\mathcal{DR}$ runs *DecryptResult* (Alg. 5) to recover $m'(X)$, extract bytes from J_i, stop at padding zero, and reconstruct d_i as a valid JSON object (or a scalar if $record_s = 1$).

Algorithm 1 *InitLedger* (chaincode)

Require: n; $record_s^{\mathcal{DW}}$; op: hint
$\log N \leftarrow \mathsf{selMinLogN}\Big(\big(n,\, 8 \cdot \big\lceil \frac{record_s^{\mathcal{DW}}}{8} \big\rceil\big)\Big)$
if $\log N = \varnothing$ **then**
  return $\perp$
end if
$bgvParams \leftarrow \mathsf{selParams}((\log N,\, op : \text{hint}))$
$D \leftarrow \mathsf{genRecords}\big((n,\, record_s^{\mathcal{DW}})\big)$
if $\exists i :\ |d_i| > record_s^{\mathcal{DW}}$ **then**
  return $\perp$
end if
$record_s \leftarrow 8 \cdot \left\lceil \dfrac{\max_i |d_i|}{8} \right\rceil$
if $\neg\mathsf{feasible}(\log N, n, record_s)$ **then**
  return $\perp$ // infeasible configuration
end if
$c \leftarrow [0, \ldots, 0] \in \mathbb{Z}_T^N$ // init coefficient vector
for $i \in [0, n-1]$ **do**
  $J_i \leftarrow \{i \cdot record_s, \ldots, (i+1) \cdot record_s - 1\}$ // slot window for d_i
  for $k = 0$ **to** $record_s - 1$ **do**
    if $k < |d_i|$ **then**
      $c[J_i[k]] \leftarrow \mathsf{byte}(d_i[k])$ // copy byte of record
    else
      $c[J_i[k]] \leftarrow 0$ // padding
    end if
  end for
end for
$m_{DB}(X) \leftarrow enc^{\text{poly}}(c) \in \mathbb{Z}_T[X]/(X^N + 1)$
$worldState \leftarrow \{m_{DB},\ n,\ record_s,\ bgvParams,\ op : D\}$
return OK

Algorithm 2 *GetMetadata* (chaincode)

Require: $\varnothing$
$n \leftarrow worldState.n$
$record_s \leftarrow worldState.record_s$
$paramsMeta \leftarrow worldState.bgvParams$
if $n = \varnothing \lor record_s = \varnothing \lor paramsMeta = \varnothing$ **then**
  return $\perp$
end if
$paramsMeta = (\log N, N, \log Q_i[], \log P_i[], T)$
$metadata \leftarrow (n, record_s, paramsMeta)$
return $metadata$

Algorithm 3 *FormSelectionVector* (client)

Require: pk; $i \in [n]$; $record_s$; N
if $i < 0 \lor i \geq n$ **then**
  return $\perp$
end if
if $n \cdot record_s > N$ **then**
  return $\perp$
end if
$J_i \leftarrow \{i \cdot record_s, \ldots, (i+1) \cdot record_s - 1\}$
$v_i \in \{0,1\}^N \leftarrow \mathbf{0}$
for $j \in J_i$ **do**
  $v_i[j] \leftarrow 1$
end for // windowed selector
$m_q(X) \leftarrow enc^{\text{poly}}(v_i)$ // polynomial encode at max level
$ct_q \leftarrow Enc_{pk}(m_q)$
$ct_q^{B64} \leftarrow enc^{\text{B64}}\big(ser^{\text{bin}}(ct_q)\big)$
return ct_q^{B64} // Base64(marshalled ciphertext)

Algorithm 4 *PIRQuery* (chaincode)

Require: ct_q^{B64}
if $ct_q^{B64} = \varnothing$ **then**
  return $\perp$
end if
$ct_q \leftarrow des^{\text{bin}}\big(dec^{\text{B64}}\big((ct_q^{B64})\big)\big)$
if m_{DB} not cached in memory **then**
  $m_{DB} \leftarrow worldState.m_{DB}$
end if
$ct_r \leftarrow Eval(ct_q, m_{DB})$
$ct_r^{B64} \leftarrow enc^{\text{B64}}\big(ser^{\text{bin}}((ct_r))\big)$
return ct_r^{B64}

IV. PERFORMANCE EVALUATION

A. EXPERIMENTAL SETUP

All experiments were executed on a local Ubuntu 24.04 host running under WSL2 on an Intel Core i5-3380M CPU (2 cores/4 threads, 2.90 GHz) with 7.7 GB of RAM and a 1 TB SSD. During evaluation, the average available memory was 6.9 GB with a 2 GB swap partition, and the root filesystem reported 946 GB of free space.

Algorithm 5 *DecryptResult* (client)

Require: ct_r^{B64}; sk; $i \in [n]$; $record_s$; n
if $i < 0$ **or** $i \geq n$ **then return** $\perp$
if $n \cdot record_s > N$ **then return** $\perp$ // sanity
$ct_r \leftarrow des^{\text{bin}}\big(dec^{\text{B64}}\big((ct_r^{B64})\big)\big)$
$u \in \mathbb{Z}_T^N \leftarrow dec^{\text{poly}}(\, m'(X)\,) \leftarrow Dec_{sk}(ct_r)$
$J_i \leftarrow \{i \cdot record_s, \ldots, (i+1) \cdot record_s - 1\}$
$b \leftarrow$ byte array // init empty buffer for record
for $j \in J_i$ **do**
  if $u[j] = 0$ **then**
    break
  end if // stop at padding zero
  b.append$(u[j])$
end for
if $record_s = 1$ **then return** $u[i \cdot record_s]$
$d_i \leftarrow dec^{UTF8}(b)$
return d_i

The software stack consisted of Go 1.24.1, Docker 27.4.0, and Docker Compose v2.31.0, hosting Hyperledger Fabric v2.5 with a single Raft orderer and LevelDB as the world-state database. Fabric's ordering service used the recommended parameters (*BatchSize.AbsoluteMaxBytes*=99 MB, *PreferredMaxBytes*=2 MB per block).

Our implementation employs the *Lattigo* v6 library [37] as the homomorphic encryption backend. Lattigo provides a Go-native implementation of the BGV scheme [41], whose security and correctness have been validated in prior literature. Accordingly, our focus is on evaluating its *practical performance within a permissioned blockchain environment*.

Client–peer communication was performed through the Fabric Go SDK [46]. The network configuration comprised a single organization with one peer per channel, sufficient for privacy evaluation since PIR execution occurs solely at endorsing peers and ordering nodes do not access the world state. Unless otherwise stated, each reported value represents the mean of 20 executions under both cold and warm cache conditions, cross-verified against peer logs for consistency.

B. CRYPTOGRAPHIC PERFORMANCE

Parameter Configuration. Table 2 summarizes the BGV parameter sets used in our evaluation, including the ring dimension N, ciphertext modulus chain $(\log Q_i, \log P_i)$, plaintext modulus T, slot allocation $record_s$, and database size n. Each configuration corresponds to one Fabric channel and maintains a feasible packing ratio as defined in Section III.

TABLE 2. Default BGV Parameter Configuration per Channel

N	$\log Q_i$	$\log P_i$	T	$record_s$	n
2^{13}	[54]	[54]	65537	128	64
2^{14}	[54]	[54]	65537	224	73
2^{15}	[54]	[54]	65537	256	128

Overall Results. Table 3 lists the measured execution time of core cryptographic operations (*KeyGen*, *Enc*, *Eval*, and *Dec*) for each evaluated ring dimension N. Table 4 summarizes the corresponding sizes of primary cryptographic artifacts, including public and secret keys, ciphertexts (ct_q, ct_r), the encoded plaintext database m_{DB}, and auxiliary metadata. Figure 5 consolidates the main cryptographic evaluation metrics: (A) end-to-end latency by algorithmic stage, (B) serialized artifact size, and (C) slot utilization ratio within the packed database polynomial m_{DB}.

TABLE 3. Execution Time of Cryptographic Operations (ms)

N	KeyGen	Enc	Eval	Dec
2^{13}	27.7	11.7	16.9	5.1
2^{14}	42.7	27.7	30.7	11.9
2^{15}	55.0	49.8	64.8	15.6

TABLE 4. Size of Main Cryptographic Artifacts (KB)

N	pk	sk	ct_q	ct_r	m_{DB}	Metadata
2^{13}	256.1	128.0	128.3	128.3	64.3	0.08
2^{14}	512.1	256.0	256.3	256.3	128.3	0.08
2^{15}	1024.1	512.0	512.3	512.3	256.3	0.08

C. BLOCKCHAIN PERFORMANCE

Blockchain performance. Figure 6 summarizes the performance of the CPIR chaincode within Hyperledger Fabric across three channels, each corresponding to a different ring size N and record configuration: (A) block size breakdown, (B) world-state (LevelDB) breakdown, (C) block vs. world-state size, (D) peer CPU utilization, (E) peer memory utilization, and (F) peer network I/O.

Chaincode Execution Timings. Figure 7 and Table 5 summarize the average server-side execution time of the main chaincode functions across different ring sizes.

TABLE 5. Average Chaincode Execution Time (ms)

N	*InitLedger*	*GetMetadata*	*PIRQuery*
2^{13}	165.24	6.16	12.53
2^{14}	187.03	8.31	18.17
2^{15}	307.87	5.94	40.36

Remark (execution paths). Transactions that modify world state (e.g., *InitLedger*) are issued via the *submit* path and committed through Raft consensus, while read-only operations (*GetMetadata*, *PIRQuery*) use the *evaluate* path, bypassing block creation and ordering. All metrics are averaged over 20 executions under identical peer configurations.

D. OVERALL SYSTEM PERFORMANCE

On-chain network behavior is reported in Table 6, which presents the average peer-side network I/O per *PIRQuery* transaction across the three channel configurations. Block

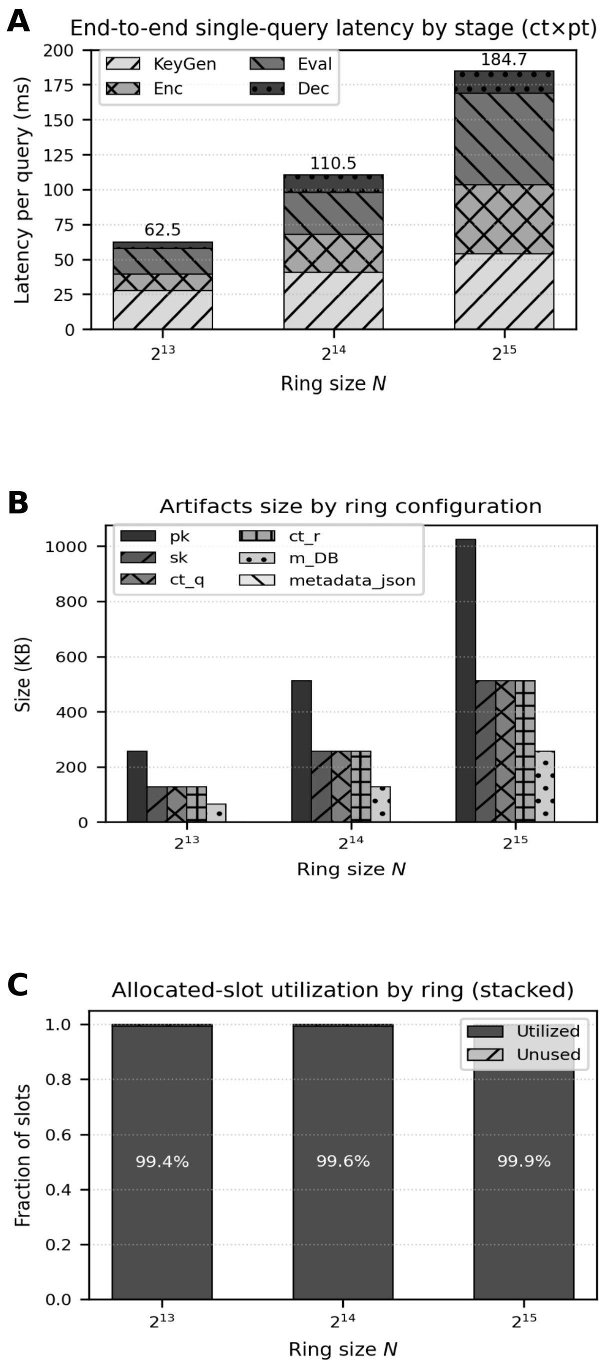

FIGURE 5. Cryptographic performance of the BGV-based CPIR system: (A) latency by algorithmic stage, (B) artifact size by ring configuration, and (C) allocated-slot utilization.

and world-state sizes are detailed in Table 7, showing the breakdown of storage components per channel. Figure 8 illustrates the peer- and client-side execution during a PIR query.

End-to-End Workflow Summary. Based on the data in Table 3 and Table 5, we now to summarize two main operational workflows in the system, involving $\mathcal{DW}$ and $\mathcal{DR}$, while $\mathcal{DO}$ is implicitly involved as the executing peer during private queries.

The first workflow corresponds to $\mathcal{DW}$ initializing the ledger through *InitLedger*, which packs and encodes the plaintext database m_{DB} into world state.

The second workflow corresponds to $\mathcal{DR}$ performing a private query by sequentially executing *GetMetadata* $\rightarrow$ *KeyGen* $\rightarrow$ *Enc* $\rightarrow$ *PIRQuery* $\rightarrow$ *Dec*. The PIR evaluation itself is performed by $\mathcal{DO}$ (endorsing peer) using ciphertext-plaintext multiplication during the evaluate transaction.

Table 8 summarizes the cryptographic and blockchain timings for both workflows, averaged across all three channel configurations.

TABLE 6. Peer network I/O per PIRQuery (avg, KB/tx)

Channel	N	NET I/O (KB/tx)
mini	2^{13}	524.62
mid	2^{14}	1042.09
rich	2^{15}	2072.88

Projected Cost for PIR Query. Among our results, for the ring size 2^{15} configuration, issuing 100/1000/10000 private queries incurs ≈207MB / 2.07 GB / 20.7 GB of peer-side network I/O with an associated chaincode evaluation time of ≈0.07/0.67/6.73 minutes, respectively, when using our protocol, as shown in Table 9 and Figure 9.

Remark (on network I/O measurement). The network I/O values in Table 6 and Table 9 were obtained from *docker stats* snapshots collected for the *peer0.org1.example.com* and *orderer0.group1.orderer.example.com* containers during each benchmark epoch. For every execution of *PIRQuery*, the benchmarking script recorded cumulative network statistics (*NET I/O*) before and after the transaction, and the Python post-processing pipeline computed per-epoch deltas ($bytes_{in/out}(t) - bytes_{in/out}(t{-}1)$) to obtain the mean transfer volume per function.

V. DISCUSSION

A. FUTURE WORK AND OUTLOOK

While this work demonstrates the feasibility of integrating CPIR directly into Hyperledger Fabric's chaincode for privacy-preserving queries, several challenges remain in performance, scalability, and deployment. Future research should address *(i)* side-channel resilience through constant-time chaincode evaluation and standardized ciphertext serialization to mitigate timing and size-based leakages. *(ii)* Scaling beyond moderate database sizes will require sublinear or sharded variants—such as preprocessing-based PIR, offline hinting, or multi-stage initialization—to reduce online complexity. Moreover, *(iii)* extending the current ciphertext–plaintext (*ct*×*pt*) evaluation to fully encrypted-database (*ct*×*ct*) computation would enable richer on-chain analytics under encryption, though it demands relinearization, rotation, and modulus-switching capabilities within Fabric's resource

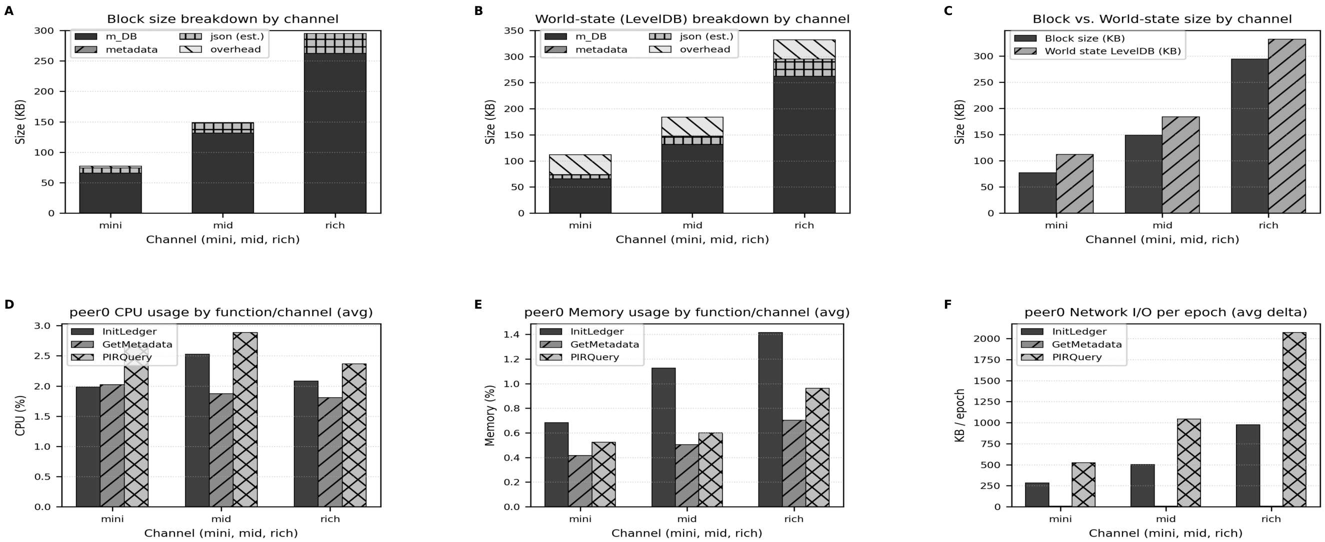

FIGURE 6. **Blockchain-side evaluation of the CPIR system across three Fabric channels. (A–C) Storage-level metrics: block and world-state composition. (D–F) Peer-level resource utilization: CPU, memory, and network I/O per function.**

TABLE 7. **Block and World-State Size per Channel Configuration**

N	n	m_{DB} (KB)	Metadata (KB)	JSON (KB)	Overhead$_{block}$ (KB)	Overhead$_{ws}$ (KB)	Block (KB)	World (KB)
2^{13}	64	65.838	0.061	8.064	3.037	38.037	77	112
2^{14}	73	131.374	0.062	16.206	1.358	36.358	149	184
2^{15}	128	262.446	0.063	32.512	0.000	36.979	294	332

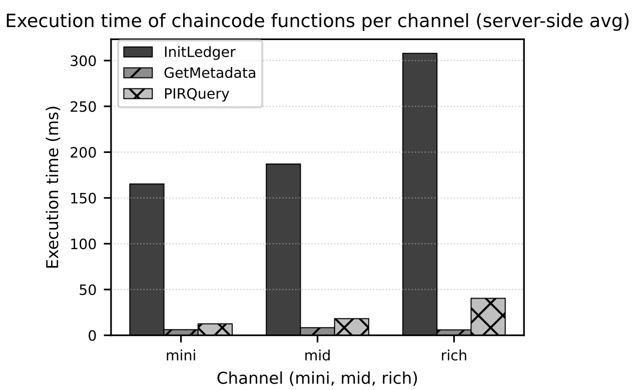

FIGURE 7. **Average chaincode execution time by function and ring size. Each bar represents the mean execution time over multiple epochs.**

TABLE 8. **End-to-End Performance Analysis (ms)**

Workflow	Cryptographic Operations (ms)	Blockchain Operations (ms)	Total Time (ms)
$\mathcal{DW}$'s Upload	0.0	220.0	220.0
$\mathcal{DR}$'s Query	82.4	30.5	112.9

limits. *(iv)* Dynamic ledger updates (*addRecord*) operation would allow incremental growth without full reinitialization, raising questions around re-encoding and key management. *(v)* Selective field-level retrieval through adaptive packing

TABLE 9. **Projected cost for multiple PIR queries at 2^{15} (peer perspective)**

# Transactions	Total Bandwidth (MB)	Total Time (min)
100	207.3	0.07
1000	2073.0	0.67
10000	20730.0	6.73

could enable varying length records and partial access control. *(vi)* Support for dual-mode operation—covering both world-state and historical ledger data—would enable privacy-preserving analytics and provenance tracking. *(vii)* Deployment optimization is another key area. In our setup, the chaincode operated in *development mode* as an external service container, which introduced duplicated gRPC transmissions for each ciphertext (ct_q, ct_r). During every *PIRQuery*, the transaction traversed two network hops—(1) client↔peer and (2) peer↔chaincode—causing each ciphertext to be sent and received twice, once per link. This resulted in roughly four full transmissions per query, explaining the measured aggregate of ≈2072 KB for $\log N$=15, consistent with the theoretical estimate 4×512 KB plus gRPC/TLS overhead. If the same chaincode were executed *in-process* within the peer container, only the client–peer exchange would occur, reducing the network footprint to approximately 2×512=1024 KB per query (about 10 GB for 10,000 queries).

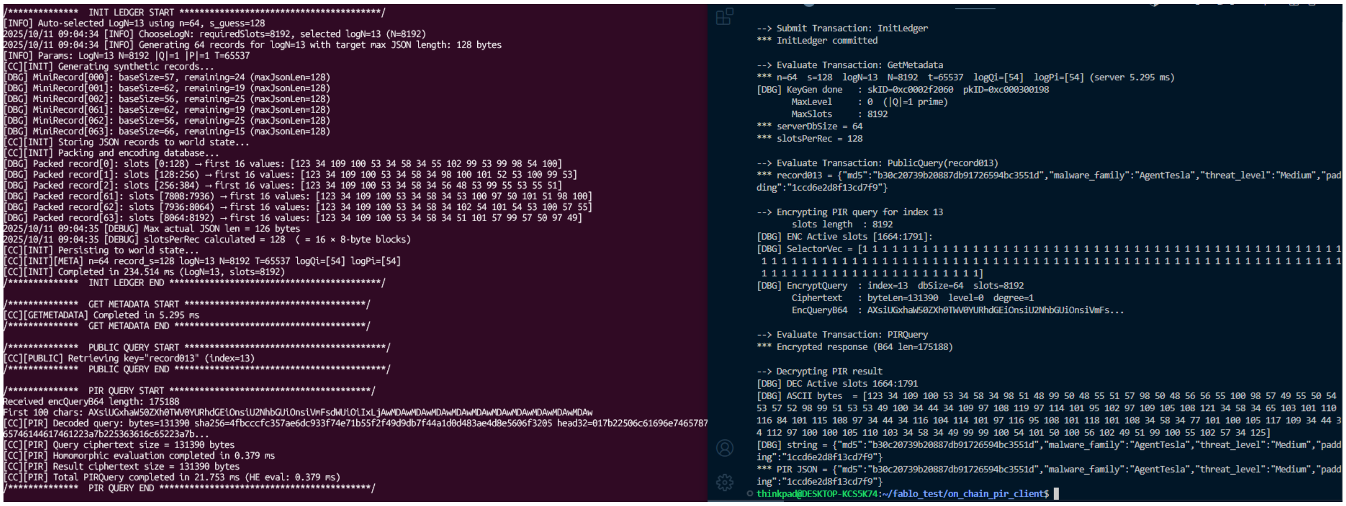

FIGURE 8. Detailed execution logs illustrating the privacy-preserving query workflow. Left: peer-side chaincode logs showing InitLedger, GetMetadata, and PIR evaluation with ciphertext-plaintext multiplication. Right: client-side logs showing metadata retrieval, query encryption, PIR query invocation, and decryption of the result.

TABLE 10. Comparison with existing privacy-preserving query systems

Work	Targeted System	Privacy Technique	Execution Location	Targeted Operation	Comm. Cost / Query (MB)	E2E Query Time (ms)	Db Size / (# of records)	Record Size / length (bytes)
[33]	HLF	OT	on-chain	read / write	N/A	600	10,000	N/A
[32]	HLF	ZKP	on-chain	read / write	N/A	N/A	10,000	N/A
[34]	Blockchain	DPF	off-chain	read	0.0002-0.0005	0.4-0.5	4-64	16-32
[35]	IPFS	HE-PIR	off-chain	read	10-15	500-16,000	1,000-1M	256,000
[36]	Bitcoin	Hybrid PIR	off-chain	read	0.65	2840	7,688-512,460	62-876
Ours	**HLF**	**HE-PIR**	**on-chain**	**read**	**1-2**	**112.9**	**16-512**	**64-512**

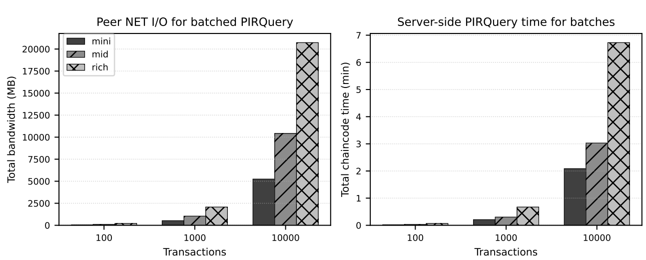

FIGURE 9. Projected cost for multiple PIR queries at 2^{15} (peer perspective)

B. RELATED WORK COMPARISON

Table 10 contrasts representative blockchain privacy frameworks mentioned in Section I, that integrate Private Information Retrieval (PIR) or related cryptographic mechanisms to achieve query privacy across different domains. The comparison focuses on six key aspects: *(i)* which target system the privacy solution is built for, *(ii)* the underlying privacy primitive (CPIR, IT-PIR, hybrid, DPF etc), *(iii)* whether privacy computation occurs on-chain (via smart contracts or chaincode) or off-chain, *(iv)* the transaction phase in which privacy logic executes (*evaluate* vs. *submit*), *(v)* typical database or world-state size, and *(vi)* reported end-to-end (E2E) query latency when available.

VI. CONCLUSION

This work demonstrated that computational private information retrieval (CPIR) can be embedded natively within Hyperledger Fabric's chaincode to enable privacy-preserving queries without modifying its consensus or ledger logic. Using the BGV homomorphic encryption scheme, query indices remain hidden from endorsing peers while retaining full endorsement and audit semantics. The prototype achieves end-to-end query latency of ≈113 ms and peer-side evaluation below 45 ms for databases up to 128 records, with key generation, encryption, and evaluation completing within 55–65 ms. Peer network monitoring indicated an average of ≈2 MB per private query due to the dual-hop gRPC communication between client, peer, and external chaincode—equivalent to ≈207 MB, 2.07 GB, and 20.7 GB for 100, 1000, and 10,000 queries, respectively. These costs can be halved by deploying the chaincode in-process within the peer, elimintating the hop and bringing cost down to ≈1 MB per query. Overall, the results confirm the practicality of executing CPIR directly within a permissioned blockchain framework, achieving millisecond-scale query performance while preserving modularity and auditability. Future directions include constant-time evaluation and standardized ciphertext serialization for side-channel resilience, sublinear or sharded CPIR for scalability, and fully encrypted (ct×ct) query processing for richer privacy-preserving analytics in distributed ledgers.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System."

[2] D. G. Wood, "Ethereum: A Secure Decentralized Generalised Transaction Ledger."

[3] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, and J. Yellick, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, ser. EuroSys '18. New York, NY, USA: Association for Computing Machinery, 2018, event-place: Porto, Portugal. [Online]. Available: https://doi.org/10.1145/3190508.3190538

[4] K. Dunnett, S. Pal, G. D. Putra, Z. Jadidi, and R. Jurdak, "A Trusted, Verifiable and Differential Cyber Threat Intelligence Sharing Framework using Blockchain," in *2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, Dec. 2022, pp. 1107–1114, iSSN: 2324-9013. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10063731

[5] P. Huff and Q. Li, "A Distributed Ledger for Non-attributable Cyber Threat Intelligence Exchange," in *Security and Privacy in Communication Networks*, J. Garcia-Alfaro, S. Li, R. Poovendran, H. Debar, and M. Yung, Eds. Cham: Springer International Publishing, 2021, pp. 164–184.

[6] Y. Allouche, N. Tapas, F. Longo, A. Shabtai, and Y. Wolfsthal, "TRADE: TRusted Anonymous Data Exchange: Threat Sharing Using Blockchain Technology," Mar. 2021, arXiv:2103.13158 [cs]. [Online]. Available: http://arxiv.org/abs/2103.13158

[7] S. Gong and C. Lee, "BLOCIS: Blockchain-Based Cyber Threat Intelligence Sharing Framework for Sybil-Resistance," *Electronics*, vol. 9, no. 3, p. 521, Mar. 2020, number: 3 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: https://www.mdpi.com/2079-9292/9/3/521

[8] P. S. o. Ethereum, "PSE Roadmap: 2025 and Beyond | PSE." [Online]. Available: https://pse.dev/blog/pse-roadmap-2025

[9] "A Blockchain Platform for the Enterprise — Hyperledger Fabric Docs main documentation." [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.5/

[10] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, "Private information retrieval," *J. ACM*, vol. 45, no. 6, pp. 965–981, Nov. 1998, place: New York, NY, USA Publisher: Association for Computing Machinery. [Online]. Available: https://doi.org/10.1145/293347.293350

[11] D. Demmler, A. Herzberg, and T. Schneider, "RAID-PIR: Practical Multi-Server PIR," in *Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security*, ser. CCSW '14. New York, NY, USA: Association for Computing Machinery, Nov. 2014, pp. 45–56. [Online]. Available: https://doi.org/10.1145/2664168.2664181

[12] I. Goldberg, "Improving the Robustness of Private Information Retrieval," in *2007 IEEE Symposium on Security and Privacy (SP '07)*, May 2007, pp. 131–148, iSSN: 2375-1207. [Online]. Available: https://ieeexplore.ieee.org/document/4223220

[13] A. Beimel, Y. Ishai, E. Kushilevitz, and J.-F. Raymond, "Breaking the O(n/sup 1/(2k-1)/) barrier for information-theoretic Private Information Retrieval," in *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, 2002, pp. 261–270.

[14] C. Devet, I. Goldberg, and N. Heninger, "Optimally Robust Private Information Retrieval," in *21st USENIX Security Symposium (USENIX Security 12)*. Bellevue, WA: USENIX Association, Aug. 2012, pp. 269–283. [Online]. Available: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/devet

[15] C. Cachin, S. Micali, and M. Stadler, "Computationally Private Information Retrieval with Polylogarithmic Communication," in *Advances in Cryptology — EUROCRYPT '99*, J. Stern, Ed. Berlin, Heidelberg: Springer, 1999, pp. 402–414.

[16] C. Gentry and Z. Ramzan, "Single-Database Private Information Retrieval with Constant Communication Rate," in *Automata, Languages and Programming*, L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, Eds. Berlin, Heidelberg: Springer, 2005, pp. 803–815.

[17] Z. Brakerski and V. Vaikuntanathan, "Efficient Fully Homomorphic Encryption from (Standard) LWE," in *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, Oct. 2011, pp. 97–106, iSSN: 0272-5428. [Online]. Available: https://ieeexplore.ieee.org/document/6108154

[18] C. Dong and L. Chen, "A Fast Single Server Private Information Retrieval Protocol with Low Communication Cost," in *Computer Security - ESORICS 2014*, M. Kutyłowski and J. Vaidya, Eds. Cham: Springer International Publishing, 2014, pp. 380–399.

[19] C. Aguilar-Melchor, J. Barrier, L. Fousse, and M.-O. Killijian, "XPIR : Private Information Retrieval for Everyone," *Proceedings on Privacy Enhancing Technologies*, 2016. [Online]. Available: https://petsymposium.org/popets/2016/popets-2016-0010.php

[20] S. Angel, H. Chen, K. Laine, and S. Setty, "Pir with compressed queries and amortized query processing," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 962–979.

[21] I. Ahmad, Y. Yang, D. Agrawal, A. El Abbadi, and T. Gupta, "Addra: Metadata-private voice communication over fully untrusted infrastructure," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, Jul. 2021, pp. 313–329.

[22] M. H. Mughees, H. Chen, and L. Ren, "Onionpir: Response efficient single-server pir," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 2292–2306. [Online]. Available: https://doi.org/10.1145/3460120.3485381

[23] S. J. Menon and D. J. Wu, "Spiral: Fast, high-rate single-server pir via fhe composition," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 930–947.

[24] A. Beimel, Y. Ishai, and T. Malkin, "Reducing the servers' computation in private information retrieval: Pir with preprocessing," *J. Cryptol.*, vol. 17, no. 2, p. 125–151, Mar. 2004. [Online]. Available: https://doi.org/10.1007/s00145-004-0134-y

[25] A. Henzinger, M. M. Hong, H. Corrigan-Gibbs, S. Meiklejohn, and V. Vaikuntanathan, "One server for the price of two: Simple and fast Single-Server private information retrieval," in *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 3889–3905. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/henzinger

[26] M. Zhou, A. Park, W. Zheng, and E. Shi, " Piano: Extremely Simple, Single-Server PIR with Sublinear Server Computation ," in *2024 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2024, pp. 4296–4314. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP54263.2024.00055

[27] B. Li, D. Micciancio, M. Raykova, and M. Schultz-Wu, "Hintless single-server private information retrieval," in *Advances in Cryptology – CRYPTO 2024*, L. Reyzin and D. Stebila, Eds. Cham: Springer Nature Switzerland, 2024, pp. 183–217.

[28] S. J. Menon and D. J. Wu, "Ypir: high-throughput single-server pir with silent preprocessing," in *Proceedings of the 33rd USENIX Conference on Security Symposium*, ser. SEC '24. USA: USENIX Association, 2024.

[29] W.-K. Lin, E. Mook, and D. Wichs, "Doubly efficient private information retrieval and fully homomorphic ram computation from ring lwe," in *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, ser. STOC 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 595–608. [Online]. Available: https://doi.org/10.1145/3564246.3585175

[30] H. Okada, R. Player, S. Pohmann, and C. Weinert, "Towards practical doubly-efficient private information retrieval," in *Financial Cryptography and Data Security*, J. Clark and E. Shi, Eds. Cham: Springer Nature Switzerland, 2025, pp. 264–282.

[31] H. Corrigan-Gibbs and D. Kogan, "Private information retrieval with sublinear online time," in *Advances in Cryptology – EUROCRYPT 2020*, A. Canteaut and Y. Ishai, Eds. Cham: Springer International Publishing, 2020, pp. 44–75.

[32] G. Kumar, R. Saha, M. Gupta, and T. H. Kim, "BRON: A blockchained framework for privacy information retrieval in human resource management," *Heliyon*, vol. 10, no. 13, Jul. 2024, publisher: Elsevier. [Online]. Available: https://www.cell.com/heliyon/abstract/S2405-8440(24)09424-6

[33] G. Kumar, R. Saha, M. Conti, and T. H. Kim, "DEBPIR: enhancing information privacy in decentralized business modeling," *Complex & Intelligent Systems*, vol. 11, no. 7, p. 290, May 2025. [Online]. Available: https://doi.org/10.1007/s40747-025-01868-y

[34] J. Xiao, J. Chang, L. Lin, B. Li, X. Dai, Z. Xiong, K.-K. R. Choo, K. Gai, and H. Jin, "Cloak: Hiding Retrieval Information in Blockchain Systems via Distributed Query Requests," *IEEE Transactions on Services Computing*, vol. 17, no. 06, pp. 3213–3226, Nov. 2024, place: Los Alamitos, CA, USA Publisher: IEEE Computer Society. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSC.2024.3411450

[35] M. Mazmudar, S. Veitch, and R. A. Mahdavi, "Peer2pir: Private queries for ipfs," in *2025 IEEE Symposium on Security and Privacy (SP)*, 2025, pp. 4438–4456.

[36] K. Qin, H. Hadass, A. Gervais, and J. Reardon, "Applying private information retrieval to lightweight bitcoin clients," in *2019 Crypto Valley Conference on Blockchain Technology (CVCBT)*, 2019, pp. 60–72.

[37] "Lattigo v6," Aug. 2024, published: Online: https://github.com/tuneinsight/lattigo. [Online]. Available: https://github.com/tuneinsight/lattigo

[38] M. Brandenburger, C. Cachin, R. Kapitza, and A. Sorniotti, "Blockchain and Trusted Computing: Problems, Pitfalls, and a Solution for Hyperledger Fabric," 2018, _eprint: 1805.08541. [Online]. Available: https://arxiv.org/abs/1805.08541

[39] J. Benet, "IPFS - Content Addressed, Versioned, P2P File System," 2014, _eprint: 1407.3561. [Online]. Available: https://arxiv.org/abs/1407.3561

[40] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptology ePrint Archive, Paper 2012/144, 2012. [Online]. Available: https://eprint.iacr.org/2012/144

[41] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "Fully homomorphic encryption without bootstrapping," Cryptology ePrint Archive, Paper 2011/277, 2011. [Online]. Available: https://eprint.iacr.org/2011/277

[42] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic Encryption for Arithmetic of Approximate Numbers," in *Advances in Cryptology – ASIACRYPT 2017*, T. Takagi and T. Peyrin, Eds. Cham: Springer International Publishing, 2017, pp. 409–437.

[43] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast fully homomorphic encryption over the torus," Cryptology ePrint Archive, Paper 2018/421, 2018. [Online]. Available: https://eprint.iacr.org/2018/421

[44] C. Peikert, "Lattice Cryptography for the Internet," in *Post-Quantum Cryptography*, M. Mosca, Ed. Cham: Springer International Publishing, 2014, pp. 197–219.

[45] C. Mouchet, J. Troncoso-Pastoriza, J.-P. Bossuat, and J.-P. Hubaux, "Multiparty Homomorphic Encryption from Ring-Learning-With-Errors," 2020, published: Cryptology ePrint Archive, Paper 2020/304. [Online]. Available: https://eprint.iacr.org/2020/304

[46] "client package - github.com/hyperledger/fabric-gateway/pkg/client - Go Packages." [Online]. Available: https://pkg.go.dev/github.com/hyperledger/fabric-gateway/pkg/client

FIRST A. AUTHOR (Fellow, IEEE) and all authors may include biographies. Biographies are often not included in conference-related papers. This author is an IEEE Fellow. The first paragraph may contain a place and/or date of birth (list place, then date). Next, the author's educational background is listed. The degrees should be listed with type of degree in what field, which institution, city, state, and country, and year the degree was earned. The author's major field of study should be lower-cased.

The second paragraph uses the pronoun of the person (he or she) and not the author's last name. It lists military and work experience, including summer and fellowship jobs. Job titles are capitalized. The current job must have a location; previous positions may be listed without one. Information concerning previous publications may be included. Try not to list more than three books or published articles. The format for listing publishers of a book within the biography is: title of book (publisher name, year) similar to a reference. Current and previous research interests end the paragraph.

The third paragraph begins with the author's title and last name (e.g., Dr. Smith, Prof. Jones, Mr. Kajor, Ms. Hunter). List any memberships in professional societies other than the IEEE. Finally, list any awards and work for IEEE committees and publications. If a photograph is provided, it should be of good quality, and professional-looking.

SECOND B. AUTHOR, photograph and biography not available at the time of publication.

THIRD C. AUTHOR JR. (Member, IEEE), photograph and biography not available at the time of publication.