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

Blind Chaincode: Enabling Computational Private Information Retrieval for Query Privacy in Hyperledger Fabric

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 endorsing peers during read operations. In Hyperledger Fabric, evaluate calls are executed by peers who observe function arguments and read-sets, creating privacy risks for organizations querying sensitive records. We address this gap by presenting the first practical integration of Computational Private Information Retrieval (CPIR) into Fabric chaincode. Our design encodes the ledger's key–value table as a plaintext polynomial and allows clients to submit encrypted selection vectors, evaluated under the Brakerski–Gentry–Vaikuntanathan (BGV) homomorphic encryption scheme. Peers return only encrypted responses, preventing index leakage while preserving normal Fabric endorsement and audit flows. We prototype the system with the Lattigo library and benchmark client-side encryption/decryption, peer-side evaluation, ciphertext size, and end-to-end query latency. Results show that single-query latencies remain practical for typical Fabric deployments, while eliminating the privacy leakage of baseline `GetState` operations. This work demonstrates the feasibility of embedding CPIR directly into permissioned blockchains and provides a foundation for future enhancements such as post-quantum schemes, zero-knowledge proofs, and sublinear retrieval.

INDEX TERMS Private Information Retrieval (PIR), Homomorphic Encryption, Hyperledger Fabric, Permissioned Blockchains, Query Privacy.

I. INTRODUCTION

A. MOTIVATION AND CONTRIBUTION

Permissioned blockchains such as Hyperledger Fabric are widely adopted for tamper-evident and auditable data management across consortiums. Their guarantees, however, primarily cover writes. In Fabric, 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. 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.

Private Information Retrieval (PIR) [1] addresses this challenge by enabling a client to retrieve an item from a database without revealing which item was requested. For a database $D = \{d_0, \ldots, d_{n-1}\}$, the client forms a one-hot selection vector v_i with a single "1" at index i. By encrypting v_i into $c_q = \mathrm{Enc}_{pk}(v_i)$ and sending it to the server, the server computes an encrypted response $c_r = c_q \cdot D = \mathrm{Enc}_{pk}(d_i)$, which decrypts to d_i under the client's secret key. This construction hides the queried index from the server. When integrated with Fabric chaincode, PIR

FIGURE 1. What endorsing peers "see" in audit records. Left: baseline *PublicQueryWithAudit* exposes the queried key (*record012*). Right: CPIR-based *PIRQueryWithAudit* exposes only an opaque encrypted selector (*EncQueryB64*), hiding the queried index.

prevents endorsing peers from linking queries to specific records, while preserving blockchain auditability for writes.

Figure 1 illustrates this contrast. A baseline query call *PublicQueryWithAudit("record012")* explicitly reveals the queried key in the audit log visible to endorsing peers. In contrast, our *PIRQueryWithAudit(encQueryB64)* logs only an opaque Base64-encoded selector, preventing the audit trail or the read-set from exposing query intent.

The main contributions of this work are:

1) **BGV-based CPIR as Fabric chaincode.** We present, to the best of our knowledge, the first fully on-chain implementation of Computational PIR (CPIR) based on the BGV scheme, integrated directly into Hyperledger Fabric chaincode.
2) **Evaluate-phase privacy demonstration.** We provide a side-by-side analysis of baseline vs. CPIR queries, showing how endorsing peers' visibility is reduced to opaque ciphertexts while preserving Fabric's normal endorsement and audit mechanisms.
3) **Prototype and benchmarks.** We implement a working system with the Lattigo library and measure client encryption/decryption, peer evaluation, ciphertext size, and end-to-end query latency, demonstrating practical performance for consortium deployments.
4) **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.

B. LITERATURE REVIEW

Privacy in permissioned blockchains has been studied from several angles, but query privacy remains underexplored.

Blockchain Privacy Mechanisms. Early efforts have emphasized access control and anonymity. Token-based authentication schemes [**?**] and access-control contracts [**?**] prevent unauthorized reads or hide participant identities. Differential-sharing frameworks [**?**] allow producers to regulate how much content is revealed. While effective at controlling *who* sees data, these mechanisms do not conceal *which* records are queried. Queries themselves remain visible to endorsing peers.

PIR and FHE Applications. A line of work has applied Private Information Retrieval (PIR) or Fully Homomorphic Encryption (FHE) to protect data access. Tan et al. [**?**] use CPIR to hide vehicular location queries. Chakraborty et al. [**?**] propose BRON, combining PIR with zero-knowledge proofs for human-resource data. Mazmudar et al. [**?**] integrate PIR with IPFS for private queries in distributed file sharing, while Hameed et al. [**?**] present DEBPIR, embedding an Oblivious Transfer–based PIR into Fabric smart contracts. These works demonstrate the feasibility of PIR in distributed settings but often rely on off-chain servers or specialized cryptographic protocols.

On-Chain CPIR Gap. Existing solutions for Fabric focus on access restriction (channels, PDC) or enclave-based confidentiality (FPC). Recent PIR-based proposals either target off-chain databases or prototype OT-based protocols. To our knowledge, no prior system has directly integrated a lattice-based CPIR scheme into Fabric chaincode. Our work closes this gap by embedding a BGV-based PIR workflow directly in Fabric's evaluate path, ensuring that endorsing peers cannot infer queried indices while preserving normal endorsement and auditability.

C. ORGANIZATION

The remainder of this paper is organized as follows: Section II surveys related work on PIR and blockchain privacy. Section III presents the system model and threat assumptions. Section IV details the design and implementation of CPIR in Fabric. Section V evaluates performance. Section VI discusses limitations and future directions, and Section VII concludes the paper.

II. PRELIMINARIES

A. TECHNOLOGY BACKGROUND

a) Fabric Native Privacy Techniques: Hyperledger Fabric separates roles among endorsing peers, committing peers, and the ordering service. Endorsing peers execute chaincode proposals and therefore observe function arguments, logs, and read-sets, making their administrators the natural adversaries for read privacy. Protecting reads in Fabric thus requires hiding query intent from endorsers.

Fabric already offers several native privacy mechanisms, each addressing a different dimension of confidentiality:

- *Separate Channels.* Multi-channel partitioning isolates ledgers across subgroups of organizations, limiting which participants observe which data. However, channel separation controls *who* sees a ledger, not *what* is accessed inside that ledger. Query intent remains visible to all endorsers of a channel.
- *Private Data Collections (PDC).* PDCs 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 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.

In summary, Fabric's native privacy tools govern data visibility and execution confidentiality. They are orthogonal to Private Information Retrieval (PIR): PDC and FPC restrict who can see data, while PIR hides what data is queried.

b) Private Information Retrieval Basics: PIR protocols enable a client to retrieve a record without revealing which record was requested. They fall into two categories:

Information-Theoretic PIR (IT-PIR). Provides unconditional privacy by distributing the database across multiple non-colluding servers. A client queries subsets of servers such that no single server learns the selection index.

Computational PIR (CPIR). Achieves privacy with a single server, relying on hardness assumptions and homomorphic encryption. For a database $D = \{d_0, \ldots, d_{n-1}\}$ and a one-hot selection vector v_i, the client computes

$$c_q = \mathrm{Enc}_{pk}(v_i), \qquad c_r = c_q \cdot D = \mathrm{Enc}_{pk}(d_i).$$

The server returns c_r, which the client decrypts as $d_i = \mathrm{Dec}_{sk}(c_r)$. Thus the queried index i remains hidden from the server.

CPIR avoids the need for multiple servers, making it attractive in blockchain settings where peers cannot be assumed non-colluding.

c) BGV Homomorphic Encryption: Our construction relies on the Brakerski–Gentry–Vaikuntanathan (BGV) scheme, a lattice-based homomorphic encryption system supporting both addition and multiplication over ciphertexts. BGV is defined over polynomial rings modulo a large ciphertext modulus and enables *batching*, where multiple plaintext elements are packed into a single ciphertext. This capability allows efficient evaluation of structured queries and underpins the polynomial database representation used in our Fabric integration.

B. Notation

We summarize the main notation used throughout the paper in Table 1. Scalars are lower-case (a), vectors are bold ($\mathbf{x}$), sets are calligraphic ($\mathcal{S}$), algorithms are sans-serif (Alg), and ciphertexts are prefixed by ct.

Remark. We write $ct_r = ct_q \cdot m_{\mathrm{DB}}$ to denote BGV ciphertext–plaintext multiplication with batching; concretely, this realizes

$$\mathsf{Enc}_{pk}(v_i) \;\cdot\; m_{\mathrm{DB}} \;=\; \mathsf{Enc}_{pk}(d_i).$$

C. Design Goals

Our design is guided by the following goals:

- **Query privacy.** Endorsing peers executing chaincode must not learn which record is being queried.
- **Auditability.** Fabric's endorsement and logging workflow should remain intact, ensuring that writes remain transparent and accountable.

TABLE 1. Notation

Symbol	Description
λ	Security parameter
n	Database size; index domain $[n] = \{0, \ldots, n-1\}$
$D = \{d_0, \ldots, d_{n-1}\}$	Database records (serialized bytes/words)
m_{DB}	Plaintext polynomial representation of D (BGV batching)
v_i	One-hot selection vector for index i
pk, sk	Public / secret keys (BGV)
$ct_q = \mathsf{Enc}_{pk}(v_i)$	Encrypted query (ciphertext)
$ct_r = \mathsf{Eval}(ct_q, m_{\mathrm{DB}})$	Encrypted response; $ct_r = \mathsf{Enc}_{pk}(d_i)$
$d_i = \mathsf{Dec}_{sk}(ct_r)$	Decrypted record d_i retrieved by client
$\mathsf{KeyGen}(\lambda) \to (pk, sk)$	Key generation
$\mathsf{Enc}_{pk}(\cdot)$, $\mathsf{Dec}_{sk}(\cdot)$	Encrypt / Decrypt
$\mathsf{Eval}(\cdot)$	Homomorphic evaluation (ct–pt multiply)
N	Ring dimension ($N = 2^{\log N}$) (BGV)
q	Ciphertext modulus (BGV)
t	Plaintext modulus (BGV)
$record_s$	Slots allocated per record (slot window size)
$record_b$	Base serialized size of a record in bytes
$record_{\mu,\log N}$	Template-specific minimum record size at security level $\log N$
$\mathcal{S}$	Allowed discrete slot sizes (implementation policy)
$\mathbf{c} = (c_0, \ldots, c_{N-1})$	Coefficient vector of the polynomial encoding (slots of m_{DB})
$\lvert\cdot\rvert$, size$(\cdot)$	Length in elements / size in bytes
$\mathcal{C}$, $\mathcal{E}$, $\mathcal{O}$, $\mathcal{P}$	Client; endorsing peers; orderer; committing peers
evaluate, submit	Fabric read / write transaction phases
$\mathcal{L}$	Leakage considered (ciphertext size, timing; defined later)

- **Practicality.** Encryption, evaluation, and decryption latencies must remain within practical limits for deployment in consortium networks.
- **Compatibility.** The protocol must run as standard Fabric chaincode without requiring protocol extensions or specialized hardware.

D. Cryptographic Primitives

We instantiate Computational Private Information Retrieval (CPIR) as a tuple of probabilistic polynomial-time algorithms

$$\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec}),$$

defined as follows:

- $\mathsf{KeyGen}(\lambda) \to (pk, sk)$: On input the security parameter λ, output a public key pk and a secret key sk.
- $\mathsf{Enc}_{pk}(v_i) \to ct_q$: Given a one-hot selection vector $v_i \in \{0,1\}^n$, produce an encrypted query ciphertext ct_q under pk.
- $\mathsf{Eval}(ct_q, m_{\mathrm{DB}}) \to ct_r$: Given an encrypted query ct_q and the plaintext database polynomial m_{DB}, homomor-

phically evaluate the product to obtain an encrypted response ct_r.
- $\mathsf{Dec}_{sk}(ct_r) \to d_i$: Using the secret key sk, decrypt the response ciphertext ct_r to recover the desired record d_i.

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

$$\mathsf{Dec}_{sk}(\mathsf{Eval}(\mathsf{Enc}_{pk}(v_i), m_{\mathrm{DB}})) = d_i.$$

a: Instantiation with BGV.

We employ the Brakerski–Gentry–Vaikuntanathan (BGV) lattice-based homomorphic encryption scheme to realize Π. BGV supports addition and multiplication on encrypted vectors and enables *batching*, where multiple plaintext slots are packed into one ciphertext. In our design, this batching is used to embed the ledger's key–value table into a single polynomial m_{DB}, allowing efficient evaluation of PIR queries inside chaincode.

III. Proposed System

A. System Model

In this article, we introduce a blockchain-based query privacy system designed for permissioned ledgers. The system enables clients to privately retrieve records while maintaining Hyperledger Fabric's endorsement and audit workflow through smart contracts. The novelty of our approach lies in the integration of computational Private Information Retrieval (CPIR) into Fabric chaincode using the Brakerski–Gentry–Vaikuntanathan (BGV) homomorphic encryption scheme. This design allows endorsing peers to execute read-only queries over encrypted inputs without learning which record was accessed. To safeguard query confidentiality and mitigate the risk of access-pattern leakage, we employ a combination of homomorphic encryption and Fabric's chaincode execution model. 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 (DO):** In our context, the Data Owners are the endorsing peers of the Hyperledger Fabric network. They maintain the plaintext polynomial representation of the ledger database m_{DB} in their world state. Upon receiving an encrypted query, the DO executes the PIR chaincode, homomorphically evaluates the query against m_{DB}, and returns only the encrypted response ct_r. The DO is modeled as *honest-but-curious*, faithfully executing the protocol but potentially attempting to infer query intent from observed arguments or logs.
- **Data Requester (DR):** The Data Requester is a client organization that wishes to privately obtain a record from the ledger. The DR generates a homomorphic encryption key pair (pk, sk), constructs a one-hot encrypted query $ct_q = \mathsf{Enc}_{pk}(v_i)$, and submits it via the PIR chaincode interface. Only the DR holds the secret key sk and is able to decrypt the response to recover the requested record d_i.
- **Ordering Service (OS):** The ordering service provides consensus by sequencing transactions into blocks. Since PIR queries are executed during the evaluate phase (read-only proposals, not committed to the ledger), the OS is not directly involved in query privacy. Its role is limited to normal write operations, ensuring global ordering and consistency of the ledger.
- **Committing Peers (CP):** These peers validate endorsed transactions and update the world state after block commitment. As with the OS, committing peers are unaffected by PIR queries but remain essential for preserving ledger integrity and synchronization across the consortium.
- **Gateway (PF):** The Gateway refers to the Fabric chaincode interface through which DRs and DOs interact. It is not a trusted third party, but rather a set of deterministic smart contracts that coordinate CPIR operations. The Gateway ensures that encrypted queries and responses flow correctly between DRs and DOs, while preventing any alteration of sensitive data. All trust-critical logic (query evaluation, ciphertext serialization, and endorsement) is encapsulated in chaincode, ensuring tamper-resistance and transparency without relying on an external intermediary.

Remark. In Hyperledger Fabric, the term "ledger" encompasses both the blockchain log of all transactions and the world state snapshot of the latest key–value pairs. Our scheme operates on the world state: the plaintext polynomial m_{DB} represents the current key–value table at the time of query. Thus PIR evaluation retrieves records from the up-to-date database snapshot, not from historical transaction logs.

B. Security Assumptions and Threat Model

Our design follows the standard *honest-but-curious* adversarial model. Endorsing peers (DOs) are assumed to correctly execute chaincode but may attempt to infer sensitive information from inputs, logs, or read-sets. We explicitly consider the following assumptions and threats:

- **Honest-but-curious endorsers.** Endorsing peers are the primary adversarial vantage point. They can observe all chaincode inputs during evaluate calls and may record ciphertexts, metadata, or execution traces. Their goal is to infer the index of the queried record.
- **Network adversaries.** External attackers may eavesdrop on ciphertexts transmitted between Data Requesters (DRs) and Data Owners (DOs). Security relies on the hardness assumptions of the BGV homomorphic encryption scheme; without the secret key sk, ciphertexts are indistinguishable from random.
- **Out-of-scope threats.** We do not address traffic analysis (e.g., repeated queries, frequency analysis) or timing side-channels (e.g., differences in execution latency).

These leakages are orthogonal to our CPIR construction and are discussed as future work.

Security objective. The system ensures that, for any query index $i \in [n]$, endorsing peers and external observers cannot distinguish which record d_i was requested. 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. 2.

1) **Ledger initialization.** Data Owners (DOs) encode the database $D = \{d_0, \ldots, d_{n-1}\}$ into a plaintext polynomial m_{DB} and store it in the Fabric world state through the InitLedger function.
2) **Metadata discovery.** Data Requesters (DRs) obtain structural parameters such as the number of records n and slots per record using lightweight chaincode calls (e.g., GetMetadata). This step reveals no sensitive content and is necessary to construct valid PIR queries.
3) **Private retrieval.** The DR generates a one-hot vector v_i for the desired index i, encrypts it into $ct_q = \mathsf{Enc}_{pk}(v_i)$, and submits it via the PIRQuery function. The DO evaluates ct_q against m_{DB} to compute $ct_r = \mathsf{Eval}(ct_q, m_{\mathrm{DB}}) = \mathsf{Enc}_{pk}(d_i)$ and returns the ciphertext.
4) **Decryption.** The DR decrypts ct_r using sk, recovering $d_i = \mathsf{Dec}_{sk}(ct_r)$. Only the requester can reconstruct the plaintext record, while endorsing peers learn nothing about which index was queried.

This design ensures that all Fabric entities retain their original roles: endorsers execute queries, the ordering service and committing peers manage writes, and clients hold cryptographic keys. The novelty is that endorsing peers now operate exclusively on encrypted selectors, making query intent indistinguishable.

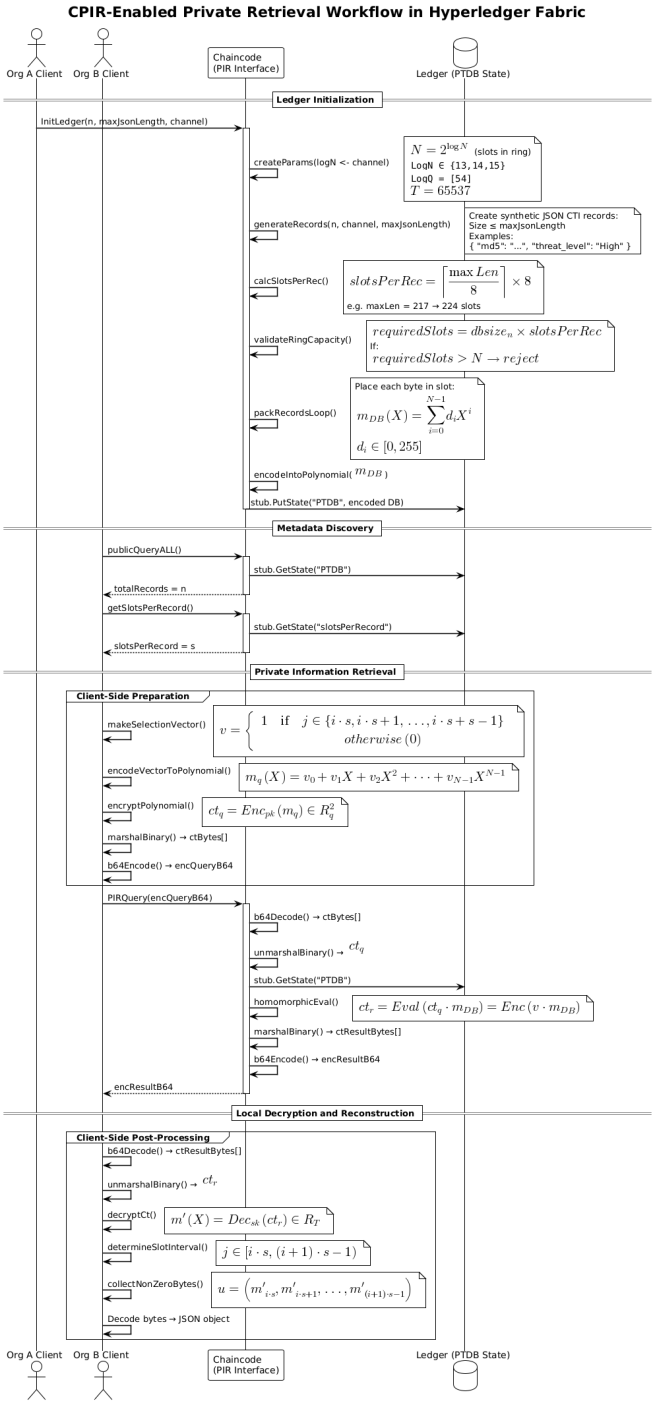

FIGURE 2. System overview. A Data Requester (DR) submits an encrypted query ct_q via the chaincode interface. The Data Owner (DO) evaluates it against the database polynomial m_{DB} and returns an encrypted response ct_r, which only the DR can decrypt.

D. Encoding and Packing Strategy

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. 3, which proceeds in four steps.

a: 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 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.

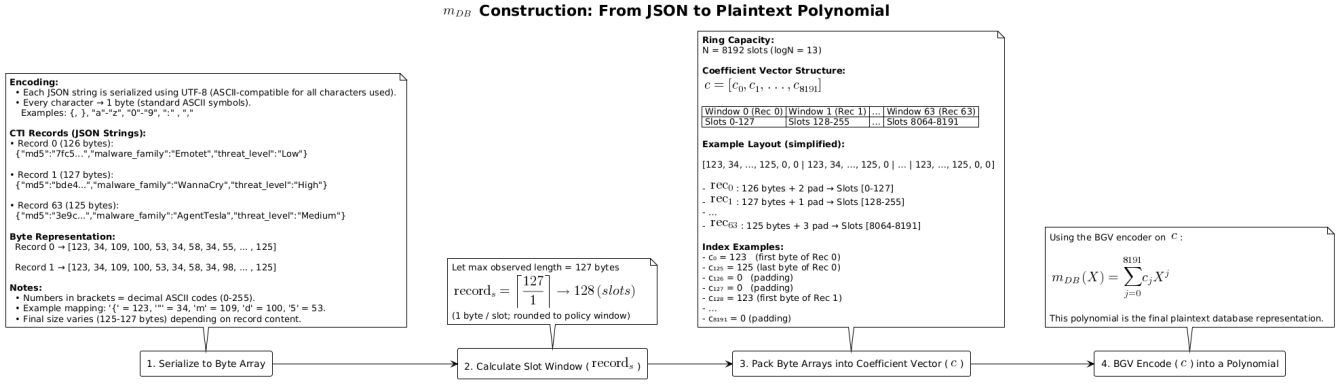

FIGURE 3. m_{DB} **construction from JSON to plaintext polynomial. Each record is serialized to bytes, mapped into a fixed slot window** $record_s$**, and packed into a coefficient vector** c**. The vector is then encoded as a BGV plaintext polynomial** m_{DB}**, which is stored in the Fabric world state.**

b: 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.

c: 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.

d: Step 4: Encode into Polynomial.

Finally, the coefficient vector d is encoded into a plaintext polynomial

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

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.

This four-step construction balances simplicity and generality. While fixed-width packing introduces some wasted slots, it guarantees consistent indexing and enables record-level PIR retrieval. Selective field-level retrieval would require adaptive packing with variable slot windows, which we leave as future work.

E. Packing 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.

a: Constraint 1: Ring capacity (global upper bound).

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.

b: Constraint 2: Record structure requirements (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.

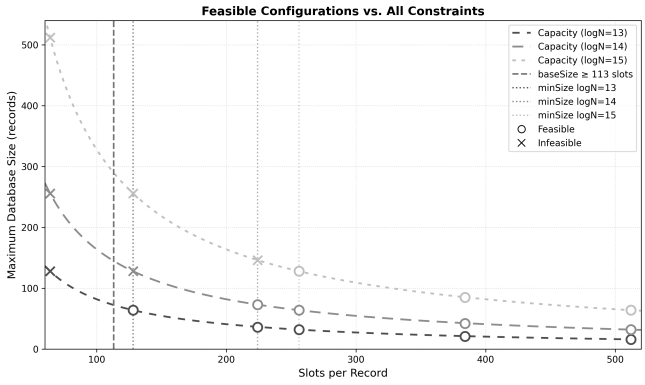

FIGURE 4. Feasible configurations under all constraints. Dashed curves: ring capacity for $\log N \in \{13, 14, 15\}$. Vertical lines: template-driven minima $record_{\mu,\log N}$ (Constraint 2). Discrete windows (Constraint 3) align with x-axis ticks. Circles mark feasible $(\log N, record_s, n)$ triples.

c: 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 $record_s \in \mathcal{S}$.

d: Constraint hierarchy and feasibility condition.

We capture feasibility using three predicates:

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

where $s = slotsPerRecord$. A configuration is feasible iff $\mathsf{Cap} \wedge \mathsf{Min} \wedge \mathsf{Disc}$.

e: Examples of constraint interaction.

- $\log N = 13$**, Mini template:** Requires $record_s \geq record_{\mu,13} \approx 128$. The smallest viable window is 128 (since 64 is insufficient). Capacity yields $n \leq \lfloor 8192/128 \rfloor = 64$.
- $\log N = 14$**, Mid template:** Requires $record_s \geq record_{\mu,14} \approx 224$. Viable windows are 224, 256, etc. With $N = 16384$, $n \leq \lfloor 16384/224 \rfloor = 73$.
- $\log N = 15$**, Rich template:** Requires $record_s \geq record_{\mu,15} \approx 256$. Windows 64, 128, 224 are invalid. With $N = 32768$, $n \leq \lfloor 32768/256 \rfloor = 128$.

f: Design implications.

This constraint system creates a natural trade-off: higher security levels ($\log N$) support larger databases but require more feature-rich record structures. The discrete allocation policy simplifies implementation while ensuring predictable performance. The experimental matrix reveals that viable configurations cluster where template requirements align with cryptographic capacity, guiding practical parameter selection.

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, where each channel is provisioned with a distinct BGV parameter set and record template.

a: Channel Mini (LogN=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 $\max_i |d_i| \leq 64$ bytes, and 16 records when $\max_i |d_i| \leq 512$ bytes.

b: Channel Mid (LogN=14).

Targets medium-sized records that include MD5 and truncated SHA-256 fields 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 (LogN=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.

d: Channel semantics.

Each channel maintains its own plaintext polynomial database m_{DB} encoded according to Alg. **??**. Fabric's world state thus contains both:

1) The *normal view*: JSON-formatted CTI records for auditability and integration with non-PIR chaincode.
2) The *polynomial view*: m_{DB}, used exclusively by the PIR chaincode for homomorphic evaluation.

This dual representation ensures that endorsing peers can still serve conventional read requests (e.g., PublicQueryALL) while PIR queries operate only over the encrypted polynomial interface.

IV. CONCLUSION

The conclusion goes here.

APPENDIX

Appendixes, if needed, appear before the acknowledgment.

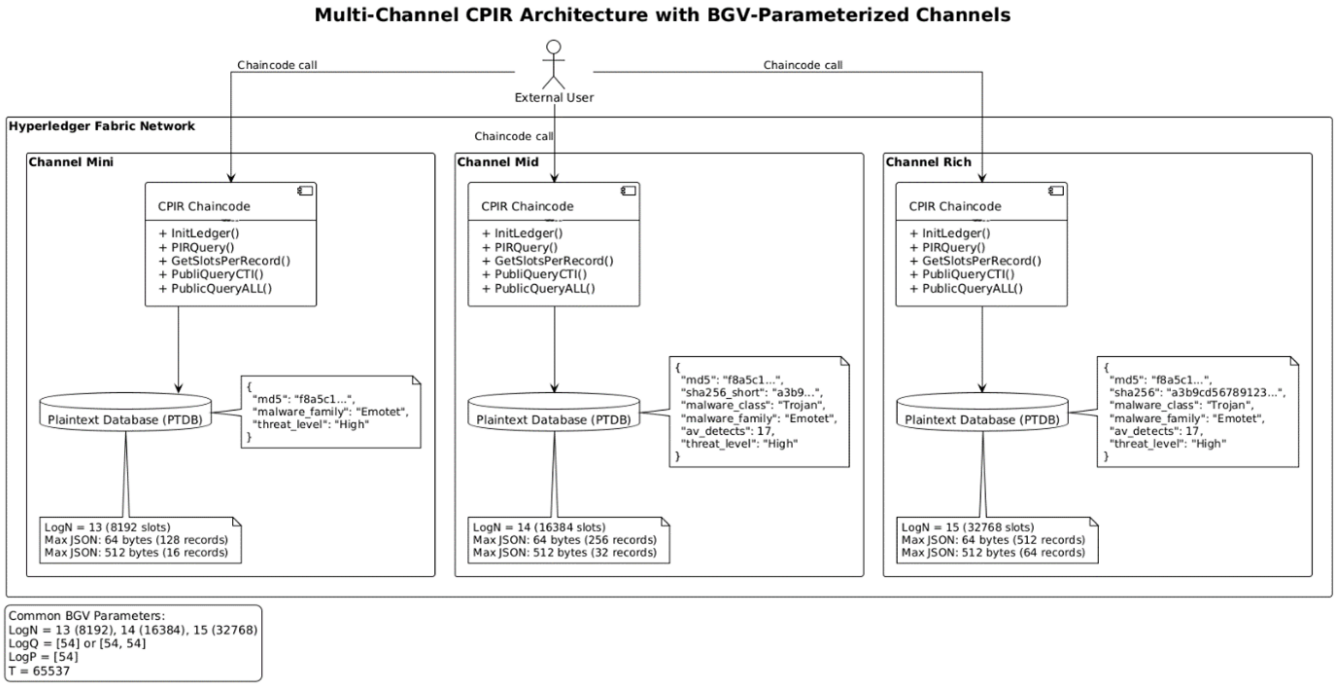

FIGURE 5. Multi-channel CPIR architecture. Each channel instantiates a separate CPIR chaincode and maintains its own m_{DB} polynomial, parameterized by $\log N$. This allows compact, mid-size, and rich CTI records to coexist under the same Fabric network.

ACKNOWLEDGMENT

The preferred spelling of the word "acknowledgment" in American English is without an "e" after the "g." Use the singular heading even if you have many acknowledgments. Avoid expressions such as "One of us (S.B.A.) would like to thank . . . ." Instead, write "F. A. Author thanks . . . ." In most cases, sponsor and financial support acknowledgments are placed in the unnumbered footnote on the first page, not here.

REFERENCES

[1] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, "Private information retrieval," *J. ACM*, vol. 45, no. 6, pp. 965–981, nov 1 1998.

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.