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 $\hat{v}_i$ with a single "1" at index i. By encrypting $\hat{v}_i$ into $c_q = \mathrm{Enc}_{pk}(\hat{v}_i)$ 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 $\hat{v}_i$, the client computes

$$ct_q = \mathrm{Enc}_{pk}(\hat{v}_i), \qquad ct_r = ct_q \cdot D = \mathrm{Enc}_{pk}(d_i).$$

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. 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 structured PIR queries inside chaincode 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.

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)
$\hat{v}_i$	One-hot selector for index i (single 1, rest 0)
v_i	Windowed selector for index i with $record_s$ contiguous ones (retrieves full record window)
pk, sk	Public / secret keys (BGV)
$ct_q = \mathsf{Enc}_{pk}(\hat{v}_i)$	Encrypted query
$ct_r = \mathsf{Eval}(ct_q, m_{\mathrm{DB}})$	Encrypted response
$d_i = \mathsf{Dec}_{sk}(ct_r)$	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)
$N = 2^{\log N}$	Ring dimension (polynomial degree of the scheme)
$\log Q_i$	Bit-lengths of primes forming modulus chain Q (ciphertext levels)
$\log P_i$	Bit-lengths of special primes P (used for key switching / relinearization)
T	Plaintext modulus (NTT-friendly prime; Lattigo `PlaintextModulus`)
$record_s$	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)
$c = (c_0, \ldots, c_{N-1})$	Coefficient vector of the polynomial encoding (slots of m_{DB})
$\lvert\cdot\rvert$, size$(\cdot)$	Length in elements / size in bytes
$\mathsf{Cap}(N, s, n)$	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 for $(\log N, n, record_s)$
$\mathcal{DO}$, $\mathcal{DW}$, $\mathcal{DR}$, $\mathcal{GW}$	Data Owner; Data Writer; Data Requester; Gateway
evaluate, submit	Fabric read / write transaction phases
$\mathcal{L}$	Leakage considered (ciphertext size, protocol timing)

C. Cryptographic Primitives

The Brakerski–Gentry–Vaikuntanathan (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. 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.

We instantiate Computational PIR 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.

- $\mathsf{Enc}_{pk}(\hat{v}_i) \to ct_q$: 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.
- $\mathsf{Eval}(ct_q, m_{\mathrm{DB}}) \to ct_r$: Given ct_q and the plaintext polynomial database $m_{\mathrm{DB}} \in R_T$, homomorphically evaluate the product to obtain an encrypted response $ct_r \in R_Q$.
- $\mathsf{Dec}_{sk}(ct_r) \to d_i$: Using the secret key sk, decrypt the response ciphertext ct_r to recover the desired record d_i.

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

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

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.

III. Proposed System

A. System Model

We introduce a blockchain-based query privacy system designed for permissioned ledgers. The system enables clients to privately retrieve from the ledger while endorsing peers can evaluate read-only queries over encrypted inputs without learning which record was accessed. The novelty of our approach lies in the integration of computational Private Information Retrieval (CPIR) into Hyperledger Fabric chaincode using the Brakerski–Gentry–Vaikuntanathan (BGV) homomorphic encryption scheme. 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 $\mathsf{KeyGen}(\lambda) \to (pk, sk)$, forms $ct_q = \mathsf{Enc}_{pk}(v_i)$, 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, GetMetadata, 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.

B. Security Assumptions and 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.
- **Out of scope.** Traffic analysis and timing side-channels; the only permitted leakage is $\mathcal{L}$ (ciphertext size and protocol timing).

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. 2.

1) **Ledger initialization.** $\mathcal{DW}$ invokes InitLedger via $\mathcal{GW}$ using submit. Chaincode derives $record_s$ from $record_b$, packs D into $c = (c_0, \ldots, c_{N-1})$, 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 $ct_q = \mathsf{Enc}_{pk}(v_i)$ and invokes PIRQuery via evaluate. $\mathcal{DO}$ computes $ct_r = \mathsf{Eval}(ct_q, m_{\mathrm{DB}})$ and returns it.
4) **Decryption.** $\mathcal{DR}$ decrypts ct_r to recover $d_i = \mathsf{Dec}_{sk}(ct_r)$.

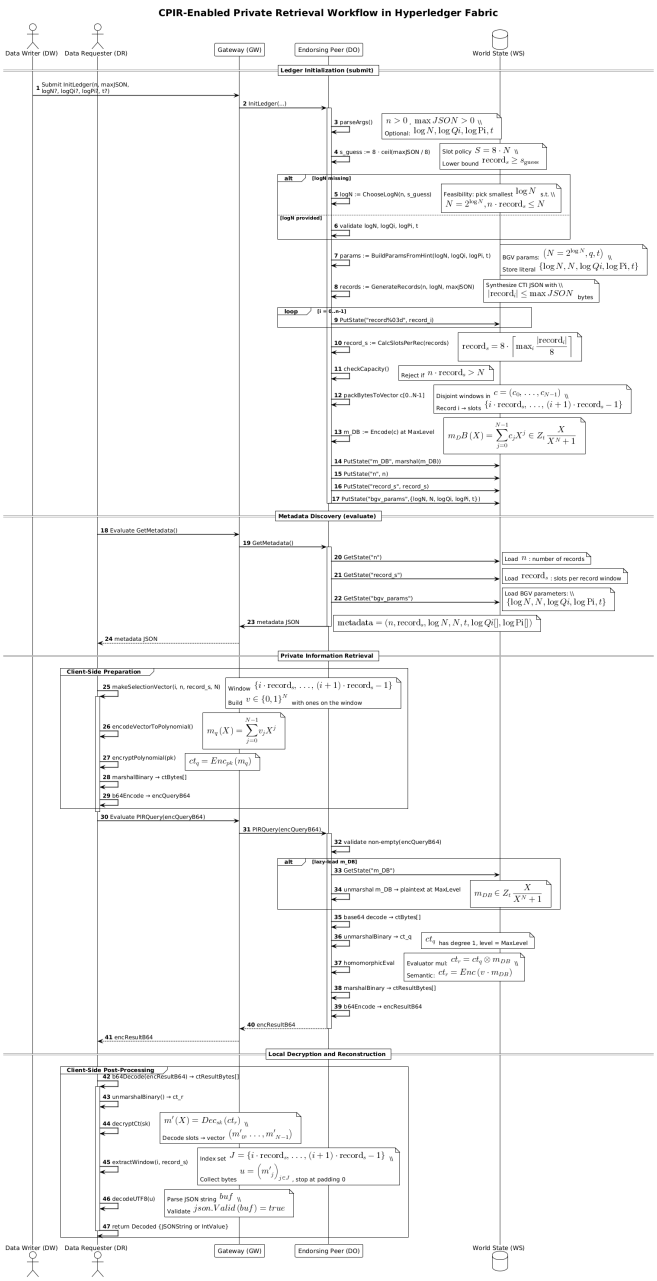


FIGURE 2. "Workflow. $\mathcal{DW}$ initializes the ledger via $\mathcal{GW}$, which triggers chaincode on endorsing peers ($\mathcal{DO}$). $\mathcal{DO}$ executes the protocol and persists state in world state (m_{DB}, metadata, JSON records). $\mathcal{DR}$ later obtains metadata, submits $ct_q = \mathsf{Encpk}(v_i)$, $\mathcal{DO}$ evaluates $ct_r = \mathsf{Eval}(ct_q, m\mathrm{DB})$ against world state, and $\mathcal{DR}$ decrypts to d_i.

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.

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.

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:

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

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. 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.

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.

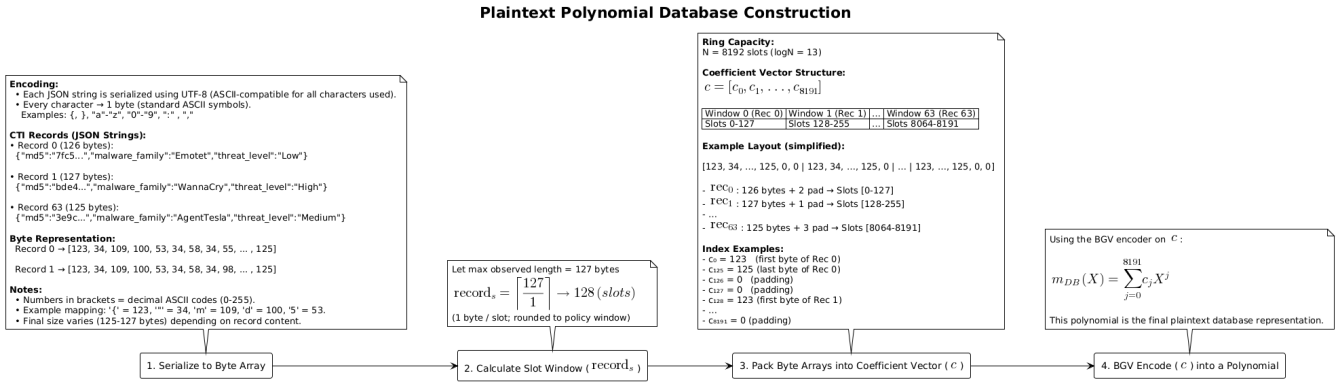


FIGURE 3. m_{DB} **construction from JSON to plaintext polynomial. Each record is serialized to bytes, mapped into a fixed slot window** $record_s$**, and packed into a coefficient vector** c**. The vector is then encoded as a BGV plaintext polynomial** m_{DB}**, which is stored in the Fabric world state.**

- **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:

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

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 $record_s \in \mathcal{S}$.

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

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

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

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

Examples. The interaction of these predicates is illustrated in Fig. 4. We provide three concrete examples below:

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

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

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

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

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

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

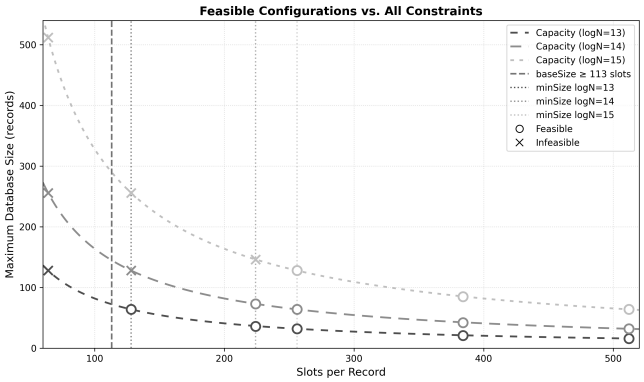


FIGURE 4. **Feasible configurations under the joint constraints** Cap**,** Min**, and** Disc**. Dashed curves show ring-capacity limits for** $\log N \in \{13, 14, 15\}$**, vertical lines mark template-driven minima** $record_{\mu,\log N}$**, and x-axis ticks correspond to discrete slot sizes** $\mathcal{S}$**. Circles indicate feasible triples** $(\log N, record_s, n)$**.**

Implications. Higher $\log N$ enlarges ring capacity and thus potential n, but also forces larger $record_s$ through richer record templates. The discrete allocation policy $\mathcal{S}$ simplifies implementation while bounding overhead. Feasible points

emerge only where template requirements align with ring capacity, guiding parameter selection in system deployment.

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. 5), 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 $\max_i |d_i| \leq 64$ bytes, and 16 records when $\max_i |d_i| \leq 512$ bytes.

b) Channel Mid ($\log N = 14$). 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 ($\log N = 15$). 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. 5, 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 individually under keys `"record%03d"` for auditability and interoperability with non-PIR chaincode.
- *Metadata*:
  - `"n"`: number of records n,
  - `"record_s"`: slots per record $record_s$,
  - `"bgv_params"`: $\{\log N, N, \log Q_i, \log P_i, T\}$.

Remark (ledger capacity). A practical concern is the maximum size of the ledger when channels store both JSON records and the polynomial m_{DB}. In Hyperledger Fabric, two layers impose constraints:

a) World state (LevelDB/CouchDB). Fabric does not set a fixed cap on the number of records n; limits are determined by disk capacity and I/O throughput. CouchDB enforces a configurable `max_document_size`, which applies to large values such as m_{DB}. Our construction stores m_{DB} as a single world-state entry, plus metadata (`"n"`, `"record_s"`, `"bgv_params"`) and optional JSON records (`"record%03d"`). Feasibility is therefore governed primarily by the size of m_{DB} and by the slot constraints in Fig. 4, not by Fabric itself.

b) Ledger history (blockchain log). Block size is limited by ordering-service parameters (`AbsoluteMaxBytes`, `PreferredMaxBytes`). However, PIR queries are executed during the evaluate phase and do not produce blocks. Only submit transactions (e.g., InitLedger or record updates) contribute to block payloads. Thus block size limits are relevant only when initializing or refreshing m_{DB}, not during PIR queries.

Implication. The effective capacity of a channel is determined by cryptographic feasibility (§4) and the maximum size of a single world-state value, rather than by Fabric's block size. For very large CTI objects, a common extension is to store them off-chain (e.g., in IPFS) and persist only their content identifiers (CIDs) in world state, while m_{DB} continues to support private retrieval of the structured fields.

G. Workflow Narrative

The complete workflow of our CPIR-enabled Fabric system consists of four stages, illustrated in Fig. 2. A Data Writer ($\mathcal{DW}$) initializes the ledger by encoding the database $D = \{d_0, \ldots, d_{n-1}\}$ into a plaintext polynomial m_{DB} and submitting it via InitLedger. Data Requesters ($\mathcal{DR}$) subsequently discover the necessary parameters by invoking GetMetadata, which exposes only structural information such as n, $record_s$, and the BGV parameter set $(\log N, \log Q_i, \log P_i, T)$. To privately obtain a record, a $\mathcal{DR}$ constructs a windowed selector v_i, encrypts it into $ct_q = \mathsf{Enc}_{pk}(v_i)$, and issues it to the PIR chaincode through PIRQuery. Endorsing peers, acting as Data Owners ($\mathcal{DO}$), perform homomorphic evaluation against the stored polynomial to return $ct_r = \mathsf{Eval}(ct_q, m_{\mathrm{DB}})$. Finally, the $\mathcal{DR}$ decrypts ct_r using its secret key sk to recover $d_i = \mathsf{Dec}_{sk}(ct_r)$ and reconstructs the original JSON object. Throughout this workflow, only opaque ciphertexts and metadata are visible to $\mathcal{DO}$, ensuring that the queried index remains hidden.

1) **DW submits initialization.** $\mathcal{DW}$ calls InitLedger via $\mathcal{GW}$ (submit) with inputs $(n, record_{s,\mathcal{DW}})$ and an optional hint $(\log N, \log Q_i, \log P_i, T)$, where $record_{s,\mathcal{DW}}$ reflects the maximum JSON size anticipated by the writer.
2) **DO validates and derives parameters.** $\mathcal{DO}$ computes a provisional slot allocation $record_{s,\mathcal{GW}} = 8 \cdot \lceil record_{s,\mathcal{DW}}/8 \rceil$. If $\log N$ is absent, the smallest $\log N$ 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 built and stored.
3) **DO prepares records.** Either ingest provided CTI records or synthesize $D = \{d_0, \ldots, d_{n-1}\}$ with $|d_i| \leq record_{s,\mathcal{DW}}$.
4) **Compute slot window.** Fix $record_s = 8 \cdot \lceil \max_i |d_i|/8 \rceil$ (discrete policy).

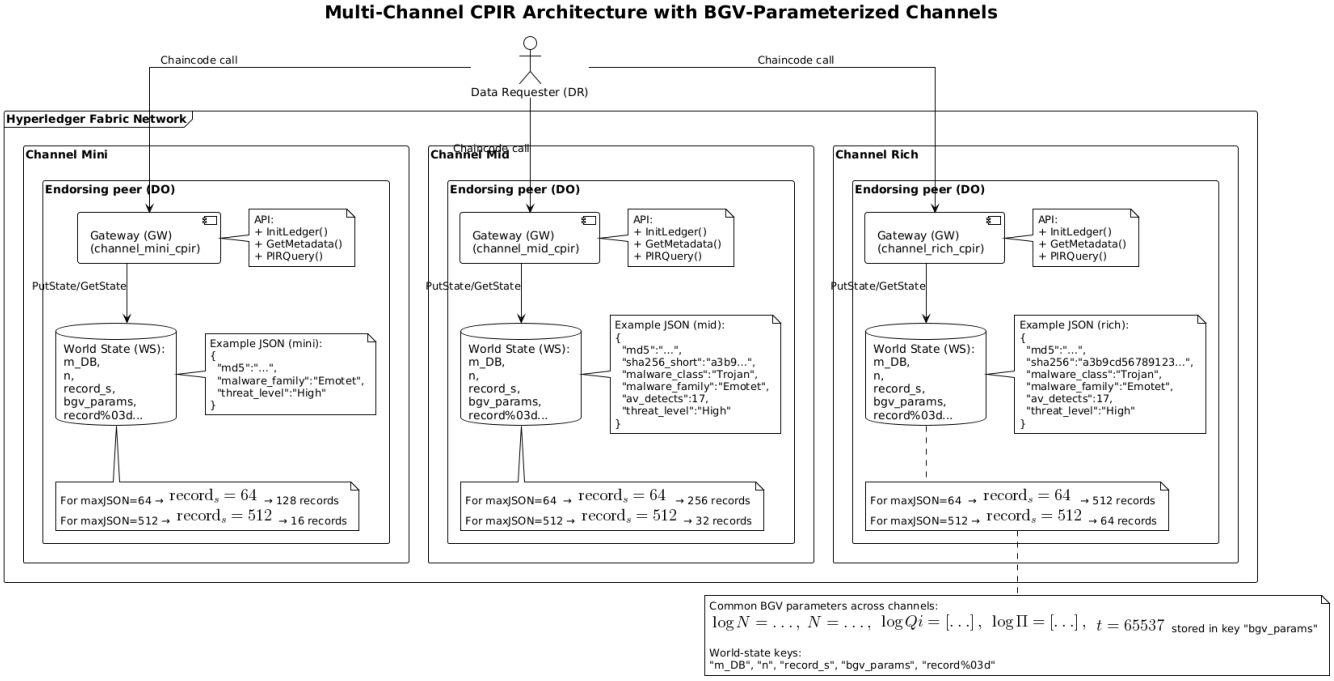


FIGURE 5. **Multi-channel CPIR architecture. Each channel instantiates a separate CPIR chaincode and maintains its own m_{DB} polynomial, parameterized by $\log N$. This allows compact, mid-size, and rich CTI records to coexist under the same Fabric network.**

5) **Feasibility check.** Reject if any predicate fails: $\mathsf{Min}(\log N, record_s)$, $\mathsf{Disc}(record_s)$, $\mathsf{Cap}(N, record_s, n)$.
6) **Pack to coefficient vector.** Assign each d_i to a disjoint slot window $J_i = \{i \cdot record_s, \ldots, (i+1)record_s - 1\}$ in $c = (c_0, \ldots, c_{N-1})$; pad with zeros.
7) **Encode polynomial and persist.** Encode $m_{\mathrm{DB}}(X) = \sum_{j=0}^{N-1} c_j X^j \in \mathbb{Z}_T[X]/(X^N + 1)$ at max level. Persist to world state under keys `"m_DB"`, `"n"`, `"record_s"`, and `"bgv_params"` $= \{\log N, N, \log Q_i, \log P_i, T\}$, plus optional `"record%03d"` entries.
8) **DR discovers metadata.** $\mathcal{DR}$ invokes GetMetadata (evaluate). $\mathcal{DO}$ returns $(n, record_s, \log N, N, T, \log Q_i, \log P_i)$.
9) **DR instantiates crypto context.** From metadata, $\mathcal{DR}$ builds parameters, runs $\mathsf{KeyGen}(\lambda) \to (pk, sk)$, and prepares encoder/encryptor.
10) **Form the windowed selector.** For index $i \in [n]$, define the window $J_i = \{i \cdot record_s, \ldots, (i+1) \cdot record_s - 1\}$. Build $v_i \in \{0,1\}^N$ with ones on J_i and zeros elsewhere.
11) **Encode and encrypt the query.** Encode v_i as $m_q(X) = \sum_{j=0}^{N-1} (v_i)_j X^j$ and encrypt $ct_q = \mathsf{Enc}_{pk}(v_i)$. Serialize and Base64-encode the ciphertext.
12) **DR issues PIR query.** $\mathcal{DR}$ calls PIRQuery(`encQueryB64`) via $\mathcal{GW}$ (evaluate).
13) **DO evaluates homomorphically.** $\mathcal{DO}$ decodes ct_q, ensures m_{DB} is loaded from world state, and computes $ct_r = \mathsf{Eval}(ct_q, m_{\mathrm{DB}})$ (ciphertext–plaintext multiply). The result is serialized and Base64-encoded for return.
14) **DR decrypts the response.** $\mathcal{DR}$ decodes and decrypts ct_r to $m'(X)$, decodes slots, and extracts bytes on J_i.
15) **Reconstruction and validation.** Trim at padding zero, decode UTF-8, validate JSON, and output d_i. If $record_s = 1$, return the scalar value directly.

Remark (levels, noise, determinism). Queries are encoded and encrypted at *max level*. Chaincode evaluates a single ct–pt multiply (no relinearization). This keeps noise growth minimal and evaluation deterministic across endorsers, which is important for Fabric endorsement. If m_{DB} is re-encoded or refreshed, $\mathcal{GW}$ still returns the same metadata blob; clients rebuild context idempotently.

IV. CONCLUSION

The conclusion goes here.

APPENDIX

Appendixes, if needed, appear before the acknowledgment.

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

Algorithm 1 InitLedger (chaincode)

Require: $n \in \mathbb{N}$; $record_s^{\mathcal{DW}} \in \{0;1\}^*$; 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
$m_{\mathrm{DB}}(X) \leftarrow \mathsf{enc}^{\mathrm{poly}}(c) \in \mathbb{Z}_T[X]/(X^N + 1)$
$worldState \leftarrow \{m_{\mathrm{DB}},\ n,\ record_s,\ bgvParams,\ op : D\}$
return OK

Algorithm 2 GetMetadata (chaincode)

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

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.

Algorithm 3 FormSelectionVector (client)

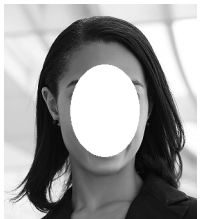
Require: pk; $i \in [n]$; $record_s$; N
if $i < 0 \ \vee\ i \geq n$ **then**
  return $\perp$
end if
if $n \cdot record_s > N$ **then**
  return $\perp$
end if
$J_i \leftarrow \{i \cdot record_s, \ldots, (i+1) \cdot record_s - 1\}$
$v_i \in \{0,1\}^N \leftarrow \mathbf{0}$
for $j \in J_i$ **do**
  $v_i[j] \leftarrow 1$
end for // windowed selector
$m_q(X) \leftarrow \mathsf{enc}^{\mathrm{poly}}(v_i)$ // polynomial encode at max level
$ct_q \leftarrow \mathsf{Enc}_{pk}(m_q)$
$ct_q^{B64} \leftarrow \mathsf{enc}^{\mathrm{B64}}\big(\mathsf{ser}^{\mathrm{bin}}(ct_q)\big)$
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
$ct_q \leftarrow \mathsf{des}^{\mathrm{bin}}\Big(\mathsf{dec}^{\mathrm{B64}}\big((ct_q^{B64})\big)\Big)$
if m_{DB} not cached in memory **then**
  $m_{\mathrm{DB}} \leftarrow worldState.m_{\mathrm{DB}}$
end if
$ct_r \leftarrow \mathsf{Eval}(ct_q, m_{\mathrm{DB}})$
$ct_r^{B64} \leftarrow \mathsf{enc}^{\mathrm{B64}}\big(\mathsf{ser}^{\mathrm{bin}}((ct_r))\big)$
return ct_r^{B64}

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
$ct_r \leftarrow \mathsf{des}^{\mathrm{bin}}\Big(\mathsf{dec}^{\mathrm{B64}}\big((ct_r^{B64})\big)\Big)$
$u \in \mathbb{Z}_T^N \leftarrow \mathsf{dec}^{\mathrm{poly}}(\,m'(X)\,) \leftarrow \mathsf{Dec}_{sk}(ct_r)$
$J_i \leftarrow \{i \cdot record_s, \ldots, (i+1) \cdot record_s - 1\}$
$b \leftarrow$ 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
  $b.\mathsf{append}(u[j])$
end for
if $record_s = 1$ **then return** $u[i \cdot record_s]$
$d_i \leftarrow dec^{UTF8}(b)$
return d_i



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.