Private Data Collections: Enabling Data Privacy in Permissioned Blockchains

Senthilnathan Natarajan
IBM Research
cendhu@gmail.com

Manish Sethi
IBM Research
Manish.Sethi1@ibm.com

Dave Enyeart
IBM
enyeart@us.ibm.com

Yacov Manevich
IBM Research
yacov.manevich@gmail.com

Artem Barger
Faculty of Information Technology,
University of Jyväskylä
Jyväskylä
artem@bargr.net

ABSTRACT

Blockchain transparency poses a barrier to enterprise adoption in domains requiring data privacy. We introduce *private data collections*, a mechanism for permissioned blockchains that decouples private data storage from the replicated ledger: private data is shared only among authorized peers, while cryptographic hashes are recorded on-chain for tamper-evidence and serializability verification. Our key theoretical contribution is **blind validation**: we prove that on-chain hashes form a *conflict-complete proxy* for private state—optimistic concurrency control (OCC) based serializability checks on hashed state detect exactly the same conflicts as checks on the underlying data, enabling the entire network to verify private transactions without accessing private data. A three-component dissemination protocol—proactive push, reactive pull, and background reconciliation—ensures data availability. The design targets any blockchain following the simulate-order-validate-commit protocol. We implement it in Hyperledger Fabric with only 3–14% throughput overhead. The mechanism has been in production since Fabric v1.2, with deployments spanning food traceability, pharmaceuticals, aerospace, financial services, and global shipping.

PVLDB Reference Format:
Senthilnathan Natarajan, Manish Sethi, Dave Enyeart, Yacov Manevich, and Artem Barger. Private Data Collections: Enabling Data Privacy in Permissioned Blockchains. PVLDB, 19(12): 4263 - 4275, 2026.
doi:10.14778/3827998.3828031

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/hyperledger/fabric.

1 INTRODUCTION

Enterprise blockchain networks are redefining organizational collaboration by establishing trust without a central intermediary. In sectors ranging from food safety [21, 39] and pharmaceuticals [35, 38] to global trade [24] and mining [23], these systems utilize a permissioned, decentralized ledger to provide a *single source of truth* for high-stakes workflows—reducing audit times from days to seconds and ensuring compliance with regulatory frameworks such as the U.S. Drug Supply Chain Security Act [37] and the EU General Food Law [13]. The value of these deployments stems from the blockchain's core properties: a transparent, immutable, and tamper-evident audit trail shared among mutually distrusting parties. However, a fundamental tension exists between this transparency and the privacy requirements of enterprise data. Sensitive information—such as negotiated pricing, HIPAA-protected medical records [19], or GDPR-governed identities [14]—cannot be fully replicated across a ledger where it is exposed to every participant, including competitors. Yet keeping such data entirely off-chain forfeits the dispute-resolution and regulatory-compliance guarantees that make blockchain valuable (§2). This creates a *privacy-auditability* deadlock: participants must prove the validity of private transactions to the network without revealing sensitive data.

Resolving this deadlock requires two properties that are individually straightforward but difficult to combine: *fine-grained privacy*, where private data remains visible only to authorized parties, and *verifiable auditability*, where on-chain cryptographic evidence allows any participant to verify integrity and trace provenance. Achieving both on a shared ledger imposes additional constraints. A single business transaction frequently updates both public and private state, so the two must commit or abort atomically; a partial commit leaves the on-chain evidence inconsistent with reality. Furthermore, the solution must deliver *enterprise-grade performance*—per-transaction overhead that does not preclude high-throughput workloads. Finally, a practical deployment constraint: the solution must be *hardware agnostic*, operating on commodity infrastructure without dependence on specialized processors whose availability, cost, or side-channel vulnerabilities narrow the set of participants that can join the network. Table 1 evaluates existing approaches against these requirements.

As the table shows, no existing approach satisfies all four requirements. *Channels* [1] require a separate ledger and consensus instance per relationship, scaling poorly and lacking cross-state atomicity. *Zero-knowledge proofs* [2, 25] provide privacy and atomicity but per-proof overhead remains prohibitive for high-throughput workloads. *TEEs* [8, 34] meet the privacy, atomicity, and performance requirements but depend on specialized hardware

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828031

Table 1: Comparison of Privacy Mechanisms

Mechanism	Fine-Grained Privacy	Cross-State Atomicity	Enterprise Performance	Hardware Agnostic
Channels	×	×	✓	✓
ZKPs	✓	✓	×	✓
TEEs (e.g., SGX)	✓	✓	✓	×
Off-chain DB	✓	×	✓	✓
Pvt. Collections	✓	✓	✓	✓

with known side-channel vulnerabilities. Off-chain databases offer privacy and performance but sacrifice cross-state atomicity (§2).

This paper formalizes the principle of *decoupled storage with on-chain cryptographic anchors* through the *Private Data Collections* (PDCs) mechanism. Our design targets blockchains that follow the *simulate-order-validate-commit* protocol (§3.2), in which transactions are first speculatively simulated on a configurable subset of peers, then ordered into blocks by a consensus service, and finally validated and committed by every peer. This pipeline creates a natural decoupling point: private plaintext is accessed only during simulation, which runs exclusively on authorized peers; the resulting cryptographic hash is then carried through ordering, validation, and commit on the replicated ledger. The core of our approach is *blind validation*: because every peer holds the hashed commitment, it can verify serializability and integrity by comparing hash versions alone—without ever accessing the underlying data. This preserves transactional atomicity across public and private states within a single shared ledger, eliminating the management overhead of channels, the computational cost of ZKPs, and the hardware dependencies of TEEs.

Because ordered blocks carry only hashes—not the private plaintext—a critical data availability challenge arises: all authorized peers must obtain the required private state before they can commit the block, yet that data never travels through the consensus-driven delivery path. We address this with a three-component dissemination protocol: a proactive gossip-based push that propagates private data among authorized peers during transaction simulation, a reactive pull at commit time for any data not yet received, and background reconciliation for peers recovering from network partitions or membership changes.

The main contributions of this paper are as follows:

- We formalize the private data collection abstraction, identify seven design requirements (atomicity, hash-data integrity, smart contract access control, authorized execution, dynamic membership, data lifecycle management, and data availability), and analyze replication protocols to determine their suitability for private data (§3).
- We design a dual-layered storage architecture (§4) built on a *blind validation* mechanism, and formally prove that on-chain hashes serve as a conflict-complete proxy for private state (Theorem 1), enabling network-wide serializability verification without exposing private data.
- We propose a three-component dissemination protocol (§4)—proactive push, reactive pull, and background reconciliation—that handles dynamic membership changes and tolerates network partitions.
- We implement PDCs in Hyperledger Fabric, in production since v1.2 and deployed by Walmart [21, 39], MineHub [23], GSBN [24], MediLedger [35, 38], Honeywell [20], and Oracle [31]. Evaluation across four industry workloads shows 3–14% overhead relative to public-only transactions.

2 MOTIVATION FOR DATA PRIVACY

Real-world blockchain deployments span diverse domains—supply chains, healthcare, trade finance—where organizations must share data with relevant parties for coordination while protecting it from unauthorized participants and maintaining auditability. The dual requirement—*share with a subset, auditable by all*—is the defining challenge. We illustrate it through food-supply-chain traceability, a domain where the tension between privacy and auditability is especially acute.

Food Traceability. In 2006, a major *E. coli* outbreak linked to bagged spinach infected 199 people across 26 U.S. states [30]. Regulators took two weeks to trace the contamination source, during which significant quantities of safe food were wasted [17]. A blockchain-based traceability system addresses this with a tamper-evident, shared record—Walmart demonstrated that blockchain could reduce backtrace time from seven days to 2.2 seconds [21, 39]. However, not all data in a food supply chain has the same visibility requirements. Invoices, pricing terms, quantities, and discount schedules are exchanged strictly between a buyer and a seller—they must remain private from other participants, including competitors transacting with the same partners.

Why private data must still be on-chain. Yet these records cannot simply be kept off-chain. When a buyer and seller later dispute the agreed price or delivery terms, the blockchain's immutable record serves as an authoritative witness that neither party can retroactively alter. Beyond conflict resolution, when an outbreak triggers an FDA investigation and companies must disclose shipment records, the regulator—and indeed any consortium member—must be able to verify that the disclosed records are genuine and unaltered, even if they never held the private data. An on-chain cryptographic anchor makes this possible: a regulator can verify a disclosed record against the ledger without the network ever having stored the plaintext—retroactive fabrication becomes impossible.

Atomicity across public and private state. Updating a public shipment record and its private invoice must succeed or fail together—if one commits without the other, the on-chain hash no longer reflects reality and the audit trail becomes untrustworthy. Food standards authorities [12, 15, 16] mandate detailed record-keeping, yet much of it is commercially sensitive. Without native atomicity, relegating private data to an external database forces organizations to maintain a *shadow* coordination layer—replicating data to authorized parties, enforcing access revocation, and reconciling consistency with the on-chain state—all independently of the blockchain, duplicating overhead the blockchain already subsumes. Integrating privacy into the ledger's native state machine is the only way to guarantee atomicity while eliminating this redundant coordination.

Privacy as a Participation Prerequisite. Beyond regulatory compliance, transparency creates an economic barrier to participation. Consortium members are often competitors, and blockchain's

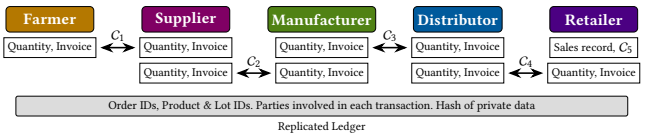

Figure 1: Private data is visible only to authorized parties; on-chain hashes provide proof of existence and traceability.

value increases with participation—yet transparency gives organizations a rational incentive to withhold private data or avoid joining altogether. The audit guarantee makes organizations willing to join despite privacy concerns: each member trusts that on-chain evidence protects the integrity of every transaction, even transactions involving private data they cannot see. Privacy is therefore not merely a compliance requirement; it is a prerequisite for consortium formation, and auditability is the trust mechanism that enables it. These requirements motivate the *private data collection* abstraction formalized next.

3 PRIVATE DATA COLLECTIONS

The mechanism rests on two complementary pillars: (1) *decoupled storage*, where private data is shared and stored only on peers owned by a defined set of authorized organizations—ensuring *privacy*; and (2) *on-chain evidence*, where the cryptographic hash of private data is recorded on the replicated ledger—ensuring *auditability*. A single transaction may modify both public and private data simultaneously, and the two must commit atomically; a partial commit renders the on-chain evidence meaningless.

Formally, a private data collection is defined as a tuple $C = (id, \mathcal{M}, \tau)$ with the following parameters:

- id: unique collection identifier
- $\mathcal{M}$: membership policy specifying the set of authorized organizations
- τ: block-to-live (BTL) retention—the number of blocks after which private data is purged, enabling compliance with data-retention regulations; $\tau = 0$ means indefinite retention

A smart contract may define multiple collections, each with independent membership policies. Additional parameters—such as the collection's endorsement policy $\mathcal{E}$ and dissemination thresholds $n_{required}$, n_{dissem}—are introduced when the formal definition is extended in §4. Figure 1 illustrates the mechanism in the food traceability setting. Five collections (C_1–C_5) are defined along participant relationships: C_1–C_4 each span two adjacent parties in the supply chain, while C_5 is private to the Retailer alone. Public data—order IDs, product lots—lands on the replicated ledger alongside a hash for each private data element; the actual quantities, invoices, and pricing terms are stored only in the relevant collection. Competitors can verify product provenance without ever seeing each other's contracts, and any participant can confirm a disclosed record's authenticity by checking it against the pre-existing on-chain hash. Because each hash is committed as part of a block, the ledger provides a tamper-evident timeline that all participants—even those that never held the private data—can trust.

3.1 Design Requirements

Table 1 captured a basic set of differentiators among privacy mechanisms. The use case in §2 and the architectural constraints of our key idea give rise to a more comprehensive set of seven design requirements, across three categories.

(1) Correctness. Integrating private data into a blockchain must not weaken the transactional guarantees. As the food traceability scenario illustrates, updating a public record and its private counterpart must be atomic—a partial commit leaves the ledger inconsistent and the on-chain hash meaningless. Beyond atomicity, the hash recorded on the ledger must faithfully reflect the off-chain data at all times; otherwise, the auditability guarantee evaporates and any party examining the ledger is working from a lie.

R1 *Transactional Atomicity and Isolation:* When a transaction modifies both public and private data, the system must guarantee atomicity and serializable isolation to maintain consistency.

R2 *Hash-Data Integrity:* The hash of the private data stored on the replicated ledger must always remain in sync with the actual private data stored in the off-chain collection, acting as a verifiable proof of existence.

(2) Access Control. Privacy guarantees are only as strong as the enforcement beneath them. In a food supply chain, pricing data must be accessible only to the buyer and seller, and access control must be embedded in the same smart contract that governs supply-chain workflows—a separate policy layer creates a seam that can be misconfigured. Equally important: private data must never reach unauthorized peers, even transiently during transaction execution.

R3 *Smart Contract Controlled Access:* Private data in a collection must only be accessible or modifiable via a smart contract, ensuring all data interactions adhere to agreed-upon business processes.

R4 *Authorized Execution Only:* Transactions involving private data must only be executed on peers owned by member organizations of the specified collections, preventing unauthorized peers from seeing or processing private data.

(3) Operational. Consortia are not static. Organizations join and leave, regulations change, and data that must be retained today may need to be purged tomorrow. A new carrier joining GSBN needs access to historical shipping documents; an organization subject to data-retention regulations may need private records purged after a mandated period without disturbing the ledger itself. Private data is also excluded from the consensus-driven block delivery—because blocks are distributed to all peers, including those not authorized to see the private data—so the system must provide an alternative path through which authorized peers can reliably obtain what they are entitled to hold.

R5 *Dynamic Membership:* Collection membership must be mutable. New members must be able to synchronize historical private data, while removed members should lose access to any data persisted after their removal.

R6 *Data Lifecycle Management:* The system must support a block-to-live (BTL) property for private data, allowing for automatic purging to comply with data retention policies or "right to forget" regulations.

R7 *Data Availability:* Since private data is excluded from the consensus-driven block delivery, the system must ensure that

all authorized peers can obtain the private data they need before committing a block, even in the presence of network partitions and peer failures.

3.2 Blockchain Architecture Requirements

We now show that private data collections require a specific class of blockchain architecture—one that natively supports selective simulation and data dissemination. In a permissioned blockchain network, a *client* submits transactions on behalf of an application. A *peer* is a node owned by a member organization that maintains a copy of the ledger, executes smart contracts, and validates transactions. An *ordering service* is a decentralized cluster of nodes that establishes the total order of transactions via a consensus protocol. Each peer belongs to exactly one organization, and organizations may operate multiple peers for redundancy. We evaluate three prevalent replication protocols, depicted in Figure 2.

Order-Execute-Commit and Execute-Order-In-Parallel & Commit. Figures 2(a) and (b) show two protocols that differ in flow: the first (Ethereum [5], Quorum [28], Hyperledger Besu [22]) orders before execution, the second (BlockchainRDBMS [29]) overlaps both. Despite this structural difference, they share a fatal property: every node must execute every transaction to maintain ledger consistency, exposing private data to all participants and breaking privacy altogether (R4). The absence of a distinct validation phase further precludes verifying private transactions without re-executing them.

Simulate-Order-Validate-Commit. Figure 2(c) separates four phases: transactions are speculatively simulated on a configurable subset of nodes, ordered by a consensus service, and then validated via authorization policy checks and optimistic concurrency control (OCC) [26] based serializability checks before commit. Hyperledger Fabric [1] implements this protocol. An *endorsement policy* governs which subset of peers simulates each transaction; each endorsing peer independently executes the smart contract and signs the resulting read-write set. The key structural property is the separation of simulation from validation: simulation can be restricted to authorized collection members (R3, R4); the validation phase can verify hashes on the replicated ledger without accessing private data (R1, R2); the gap between simulation and ordering provides a natural insertion point for peer-to-peer dissemination (R5, R7); and because execution outcomes are decoupled from the immutable ledger, private data can be stored and purged independently (R6).

4 DESIGN OF PRIVATE DATA TRANSACTIONS

The simulate-order-validate-commit protocol, as argued in the previous section, provides the right foundation for private data. We now describe how private data transaction processing is integrated within it. The design centers on three ideas: *partitioned state management* that splits state into public, hashed, and private domains; *blind validation* that uses the hashed partition as a conflict-complete proxy, enabling serializability checks without accessing private data; and *decoupled dissemination* that separates private data distribution from block ordering, with proactive push, reactive pull, and background reconciliation ensuring eventual availability. We assume a key-value data model for both public and private state, where each state entry is a $(key, value)$ pair identified by a unique key within a namespace, and each key carries a version that is updated whenever the key is written by a committed transaction. Figure 3 provides an end-to-end overview of the private data collection transaction flow, with numbered steps spanning all four phases; Peer 1 and Peer 2 are members of collection C_1, while Peer n is not. The following subsections detail each component.

4.1 Partitioned State Management

Integrating private data into a shared ledger without exposing it to unauthorized peers during validation is the central challenge. The solution is to partition the state database into three logical domains. Each smart contract is allocated a dedicated *namespace* within these domains; access and modification within a namespace are governed by an endorsement policy (§3.2) specifying the set of authorized peers whose signatures are required to validate state changes. The three partitions for each namespace are:

- *Public State:* Accessible to all nodes; contains the standard replicated ledger state.
- *Hashed State:* Replicated across all nodes; contains cryptographic hashes of private keys and values, providing the evidence required for tamper-detection and serializability checks.
- *Private State:* Isolated to authorized peers; contains the actual private key-value pairs, which are the pre-images of the hashed state. Modifications are governed by the collection's own endorsement policy $\mathcal{E}$, which supersedes the smart contract's default endorsement policy. In practice, $\mathcal{E}$ is defined over a subset of (or all of) the collection's membership $\mathcal{M}$, since only members possessing the private data can meaningfully endorse transactions.

Public State and Hashed State are consistent across all peers in the network, providing a globally uniform view of the ledger. Private State, by contrast, varies from peer to peer depending on which collections each peer's organization belongs to—a peer only maintains private state for collections in which it is authorized. By confining all private data interactions to smart contract namespaces, this partitioning satisfies **R3** (Smart Contract Controlled Access); the replicated Hashed State that mirrors Private State reinforces **R2** (Hash-Data Integrity).

4.2 Transaction Simulation

The simulation phase (steps 1–4 in Figure 3) begins when a client submits a transaction proposal to multiple endorsing peers to collect adequate endorsements as per the endorsement policies defined for the smart contract and the private data collections involved. Restricting execution to authorized peers directly satisfies **R4** (Authorized Execution Only). Private data enters the system through a *transient data* field—present during execution but never becoming part of the replicated ledger.

The goal of the simulation phase is to collect read-write sets. Reads capture the data version read by the smart contract to make decisions on the proposed changes (i.e., writes). These read-write sets are subsequently used by optimistic concurrency control (OCC) based serializability checks [26] in the validation phase. Since state is partitioned into public, hashed, and private domains (§4), the simulation produces three corresponding read-write sets—one per partition: a *public read-write set* for replicated ledger state, a *hashed*

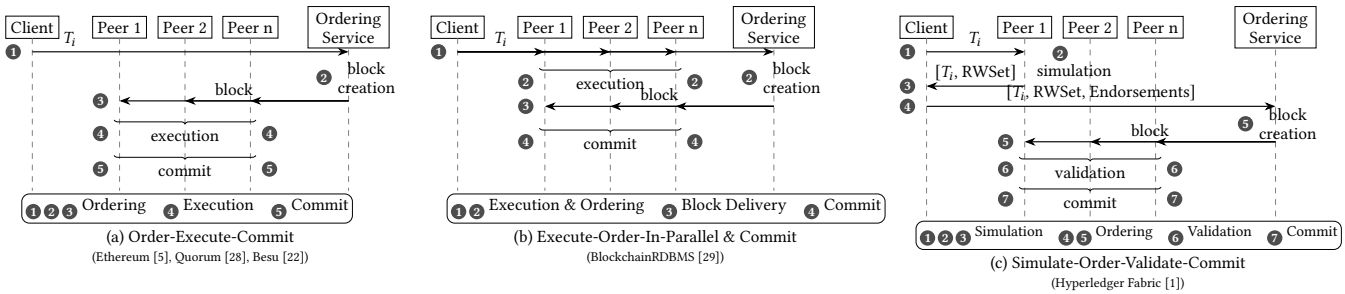

Figure 2: Blockchain replication protocols.

Table 2: Smart-contract APIs for public and private data operations, with examples illustrating simulation and blind validation.

API	*State Effect*	*Read-/Write-Set Effect*
`GetState`(k)	Returns value of k from public state	Records (k, version) in public read-set
`PutState`(k, v)	—	Records (k, v) in public write-set
`DelState`(k)	—	Records delete of k in public write-set
`GetPrivateData`(c, k)	Returns value of k from collection c	Records ($H(k)$, version) in hashed read-set
`GetPrivateDataHash`(c, k)	Returns $H(v)$ for key k in collection c	Records ($H(k)$, version) in hashed read-set
`PutPrivateData`(c, k, v)	—	Records (k, v) in private write-set; ($H(k)$, $H(v)$) in hashed write-set
`DelPrivateData`(c, k)	—	Records delete of k in private write-set; delete marker in hashed write-set
`PurgePrivateData`(c, k)	Permanently removes k from local storage	Like `Del`, but purges history to comply with "right to forget" policies

Walkthrough: Simulation and Blind Validation. T_i executes: `GetState`(k_1) → `PutState`(k_1, v_2) → `GetPrivateData`(c_1, k_2) → `GetPrivateData`(c_2, k_3) → `PutPrivateData`(c_1, k_2, v_3) → `PutPrivateData`(c_2, k_3, v_4). Before T_i reaches validation, T_j commits and updates k_2 in c_1 (version v5 → v8). The validator detects the stale read on $H(k_2)$ *without accessing the private value*—blind validation.

Phase	**Public (k_1)**	**Hashed (c_1, $H(k_2)$)**	**Hashed (c_2, $H(k_3)$)**	**Private (c_1, k_2)**	**Private (c_2, k_3)**	**Result**
1. Initial disk	value:v_1, ver:v1	value:$H(v_5)$, ver:v5	value:$H(v_5)$, ver:v5	value:v_5	value:v_5	—
2. Simulate T_i	R:v1, W:v_2	R:v5, W:$H(v_3)$	R:v5, W:$H(v_4)$	W:v_3	W:v_4	RW-sets captured
3. T_j commits	—	**ver:v5→v8**, value:$H(v_8)$	—	value:v_8	—	Disk updated
4. Validate T_i	R(v1)=Disk(v1) ✓	R(v5)≠Disk(v8) ✗	R(v5)=Disk(v5) ✓	No access	No access	T_i **invalid**

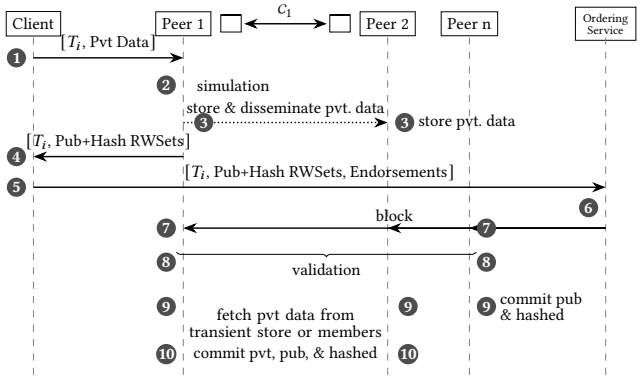

Figure 3: End-to-end private data collection transaction flow. Steps ❶–❹: simulation and dissemination, ❺–❼: ordering, ❽: validation, ❾–❿: commit and retrieval.

read-write set containing cryptographic hashes of private keys and values, and a *private write-set* per collection involved in the transaction. Note that there is no private read-set; private reads are recorded in the hashed read-set via their hash versions to enable blind validation. A separate private read-set would force member and non-member peers to follow divergent validation paths—members checking private versions while non-members could not—breaking the uniform validation guarantee that lets every peer independently reach the same commit decision.

Upon receiving the proposal, each endorsing peer executes the smart contract using the APIs defined in Table 2. Public data operations (`GetState`, `PutState`, `DelState`) populate the public read-write set. Private data operations populate the hashed and private partitions: `GetPrivateData`(c, k) records ($H(k)$, version) in the hashed read-set; `PutPrivateData`(c, k, v) performs *dual-recording*, writing (k, v) to the private write-set and simultaneously recording $(H(k), H(v))$ in the hashed write-set, satisfying **R2** (Hash-Data Integrity). `DelPrivateData` and `PurgePrivateData` similarly produce corresponding entries in both the private and hashed write-sets. Table 2 illustrates the process end-to-end: transaction T_i calls `GetState`(k_1) followed by `PutState`(k_1, v_2), producing a public read-set entry (k_1, v1) and a public write-set entry (k_1, v_2). It then calls `GetPrivateData`(c_1, k_2) and `GetPrivateData`(c_2, k_3), which record ($H(k_2)$, v5) and ($H(k_3)$, v5) in the hashed read-set. Finally, `PutPrivateData`(c_1, k_2, v_3) and `PutPrivateData`(c_2, k_3, v_4) each perform dual-recording: (k_2, v_3) and (k_3, v_4) go to the private write-sets, while $(H(k_2), H(v_3))$ and $(H(k_3), H(v_4))$ go to the hashed

write-set. When a transaction reads and writes both public and private state, the endorsing peers must satisfy both the smart contract's endorsement policy (for public writes) and the collection's policy $\mathcal{E}$ (for private writes).

Post-Simulation: Transient Storage and Dissemination. Since transaction order is determined only after simulation completes, committing private data to permanent storage at this point would risk corrupting committed state with transactions that are later invalidated. Peers therefore hold private write-sets in a *transient store* until the transaction commits, satisfying **R1** (Transactional Atomicity and Isolation). As described in Algorithm 1, after simulation the endorsing peer immediately pushes each private write-set to a subset of authorized collection members. The endorser does *not* return its signed response to the client until at least $n_{required}$ peers have acknowledged receipt. Blocking on these acknowledgments guarantees that the data exists on multiple peers before the client can submit the transaction for ordering. After sending the endorsement response, a background gossip task continues pushing the write-set to the remaining peers up to n_{dissem} total, spreading coverage without adding to client-observed latency. We separate $n_{required}$ from n_{dissem} so that operators can tune the two concerns independently. Setting n_{dissem} aggressively ensures most collection members receive the data proactively, which minimizes commit-time reactive pulls; keeping $n_{required}$ small protects endorsement latency. When the endorser cannot obtain $n_{required}$ acknowledgments before the timeout, it returns a failure to the client, who may retry with different endorsers. We deliberately do not retry dissemination at the endorser: keeping the endorsement critical path simple avoids unbounded retry delays and pushes fault handling to the client, where it belongs.

Algorithm 1 Proactive dissemination protocol.

Input: Transaction tx with private write-sets

At Endorsing Node e (after simulation):

1. Store all private write-sets for tx in local transient store
2. **for each** collection c in tx's private write-sets **do**
   - $W_c \leftarrow$ collection c's private write-set
   - *// Blocking phase: wait for $n_{required}$ ACKs before responding*
   - $P_{req} \leftarrow$ SelectAuthorizedPeers$(c.\mathcal{M}, c.n_{required})$
   - Push $(tx.id, c.id, W_c)$ to all $p_i \in P_{req}$
   - Wait until $|\{\text{org}(p_i) : p_i \text{ ACKed}\}| \geq c.n_{required}$, or timeout
   - **if** timeout **then** return dissemination failure to client
3. Return signed endorsement response to client
4. **for each** collection c in tx's private write-sets **do**
   - *// Background phase: spread to remaining peers up to n_{dissem}*
   - $P_{dissem} \leftarrow$ SelectAuthorizedPeers$(c.\mathcal{M}, c.n_{dissem}) \setminus P_{req}$
   - Push $(tx.id, c.id, W_c)$ to all $p_i \in P_{dissem}$ (non-blocking)

At Receiving Peer p_i:

1. Verify sender e is authorized for collection c
2. Verify ledger height of tx is within acceptable range
3. Store $(tx.id, c.id, W_c)$ in local transient store
4. Send ACK to endorser e

4.3 Transaction Ordering

Once the client collects sufficient endorsements, it submits the transaction—containing the public and hashed read-write sets, but not the private write-sets—to the ordering service (steps 5–7 in Figure 3). The ordering service is a black box with respect to transaction content: it never sees private data. Its sole responsibility is to order transactions into blocks using consensus (CFT or BFT), and the block order determines the commit order. Its functionality is unchanged regardless of whether a transaction involves public data, private data, or both.

4.4 Validation and Commit

Validation. Validation (step 8 in Figure 3) proceeds in two sequential checks for each transaction in a block. First, each transaction's endorsement signatures are verified against the smart contract's endorsement policy for public writes and the collection's endorsement policy $\mathcal{E}$ for private writes. A transaction that involves both public and private state must satisfy both policies. Any transaction that fails this check is immediately marked invalid and skipped, satisfying **R4** (Authorized Execution Only). Endorsement policy checks across transactions in a block can run in parallel, since they depend only on the signatures attached to each transaction and not on mutable state. Transactions that pass endorsement validation advance to OCC serializability checks [26], which compare the read-set versions recorded at simulation time against the current committed versions in the Public State and Hashed State. A transaction is serializable if no key in its read-set was modified by an intervening committed transaction. OCC validation proceeds sequentially in block order, since validating one transaction may update state versions visible to subsequent ones. A serialization graph can alternatively enable parallel validation of non-conflicting transactions [27, 36]. Every write to a private key k atomically updates its hash $H(k)$ in the replicated Hashed State, so hashed version identifiers are sufficient to detect all conflicts on private data without exposing the data itself. *Blind validation* follows directly: every peer can verify serializability of private state changes without ever seeing the private data.

Walkthrough. Continuing the Table 2 example: before T_i reaches validation, another transaction T_j commits and updates k_2 in c_1, advancing its hashed version from v5 to v8. At validation time, the validator compares T_i's recorded read versions against current disk state: public key k_1 still has version v1 (✓), hashed key $H(k_3)$ in c_2 still has version v5 (✓), but hashed key $H(k_2)$ in c_1 now has version v8, which does not match T_i's recorded version v5 (✗). The validator detects the stale read and marks T_i as invalid—all without accessing any private value. We formalize this correctness claim after describing the commit procedure below.

Commit. For each valid transaction, the committer retrieves the corresponding private write-sets from the local transient store. If the data is unavailable locally, the committer triggers a reactive pull (Algorithm 2): it first targets the transaction's endorsers, who are guaranteed to possess the write-sets, then falls back to the gossip network. Each received write-set is verified against the on-chain hash before acceptance, preventing forged data injection (§5). Validated private data is moved to the permanent *private data store*, which retains all historical write-sets indexed by block number and

transaction index to support synchronization when new members join. If reactive retrieval fails within the timeout, the peer commits the block anyway to preserve liveness, satisfying **R7** (Data Availability). Missing entries are recorded as *eligible missing data*—block number, transaction index, collection identifier, and hashed key-value pairs—so that background reconciliation can retry (§4.5); the stored hashes allow later verification. Until reconciliation completes, `GetPrivateData` for a missing key returns an error. Private data for collections in which the peer is *not* a member is tracked as *ineligible missing data*; these are never fetched but retained so that if the organization later joins, they can be reclassified to trigger synchronization, supporting **R5** (Dynamic Membership). The reverse transition is handled during membership updates (§4.5).

Algorithm 2 Reactive retrieval protocol.

Input: Block B with hashed write-sets, local transient store

At Committing Peer p (before commit):

1. **for each** transaction $tx \in B$ **do**
   if tx fails endorsement policy **then skip**
2. **for each** collection c referenced in tx's hashed write-set **do**
   if $p \in c.\mathcal{M}$ **and** c's private write-set $\notin$ transient store
   Add digest $(tx.id, c.ns, c.id, B.num, tx.idx)$ to MissingSet
   // $c.ns$: smart contract namespace owning collection c
3. **if** MissingSet = $\emptyset$ **then** proceed to commit
4. $E \leftarrow$ endorsers of tx *// priority targets*
5. Send pull request with MissingSet digests to E
6. **for each** private write-set W_c received **do**
   Verify $H(W_c)$ matches on-chain hash; if valid, store W_c
7. **if** MissingSet $\neq \emptyset$ **then**
   Broadcast to gossip network; retry with fixed-interval sleep
8. Proceed to commit (missing data tracked for reconciliation)

Proof of Validation Correctness. We now formalize the correctness of blind validation. We adopt the standard OCC serializability criterion [26]: a transaction T_i is *serializable* with respect to a committed history if no key in T_i's read-set was modified by any transaction T_j that committed after T_i began—equivalently, the read-set versions recorded at simulation time must match the current committed versions at validation time. Let $\mathcal{S}_P$ denote the Private State, $\mathcal{S}_H$ the Hashed State, and $\mathrm{ver}(s, k)$ the version of key k in state s. We call $\mathcal{S}_H$ a *conflict-complete proxy* for $\mathcal{S}_P$ if every OCC conflict in $\mathcal{S}_P$ is detectable in $\mathcal{S}_H$ and vice versa—that is, the two states agree on which transactions conflict, so validating against $\mathcal{S}_H$ is equivalent to validating against $\mathcal{S}_P$.

Theorem 1 (Conflict-Complete Proxy). *Assuming H is a collision-resistant hash function (so that $H(k_1) = H(k_2) \implies k_1 = k_2$), the Hashed State $\mathcal{S}_H$ is a conflict-complete proxy for the Private State $\mathcal{S}_P$: for any transaction T_i involving private data, an OCC conflict exists in $\mathcal{S}_P$ if and only if a corresponding conflict exists in $\mathcal{S}_H$. Formally, for every private key k in collection c:*

$\mathrm{ver}(\mathcal{S}_P, k) \neq \mathrm{ver}_{\mathrm{sim}}(T_i, k) \iff \mathrm{ver}(\mathcal{S}_H, H(k)) \neq \mathrm{ver}_{\mathrm{sim}}(T_i, H(k))$

Proof. We first establish a key invariant. By construction, every `PutPrivateData`(c, k, v) atomically produces both a private write-set entry (k, v) and a hashed write-set entry $(H(k), H(v))$. The commit procedure assigns both entries the same version identifier (block number, transaction index). `DelPrivateData` and `PurgePrivateData` similarly produce corresponding hashed entries. Thus, $\mathrm{ver}(\mathcal{S}_P, k)$ and $\mathrm{ver}(\mathcal{S}_H, H(k))$ are updated in lockstep: $\mathrm{ver}(\mathcal{S}_P, k) = \mathrm{ver}(\mathcal{S}_H, H(k))$ after every committed transaction (*Write Bijection*). Furthermore, every `GetPrivateData`(c, k) records $\mathrm{ver}(\mathcal{S}_H, H(k))$ in the hashed read-set, which by Write Bijection equals $\mathrm{ver}(\mathcal{S}_P, k)$ at simulation time. We now prove each direction.

($\Rightarrow$) Suppose a conflict exists in $\mathcal{S}_P$: some transaction T_j committed after T_i's simulation and changed $\mathrm{ver}(\mathcal{S}_P, k)$. By Write Bijection, T_j's commit also advanced $\mathrm{ver}(\mathcal{S}_H, H(k))$ by the same amount. Since the hashed read-set recorded the pre-T_j version, the OCC check on $\mathcal{S}_H$ detects the mismatch.

($\Leftarrow$) Suppose a conflict exists in $\mathcal{S}_H$: $\mathrm{ver}(\mathcal{S}_H, H(k))$ differs from the version recorded in T_i's hashed read-set. By Write Bijection, $\mathrm{ver}(\mathcal{S}_H, H(k))$ can only change when a transaction commits a write to private key k, which simultaneously advances $\mathrm{ver}(\mathcal{S}_P, k)$. Therefore, $\mathrm{ver}(\mathcal{S}_P, k)$ has also changed since simulation, confirming a conflict in $\mathcal{S}_P$.

Since both directions hold, blind validation produces neither false negatives (missed conflicts) nor false positives (spurious aborts).

4.5 Membership and Data Lifecycle

Reconciliation and Dynamic Membership. The background reconciliation service periodically retries fetching eligible missing data recorded during commit, using the same hash-based verification as Algorithm 2. When a new member is added to a collection's policy $\mathcal{M}$, the peer detects the change during block commit and reclassifies all *ineligible missing data* for that collection as *eligible*, triggering reconciliation to synchronize historical write-sets. Removal is symmetric: the departing peer reclassifies its *eligible missing data* as *ineligible*, stopping reconciliation. It retains private data committed before removal—preserving auditability—but the updated policy excludes it from future dissemination and retrieval. The departing peer can still serve pull requests from current members for historical data it holds, since on-chain hashes let the requester verify authenticity regardless of the source's current membership. If the organization is later re-added, ineligible entries revert to eligible, triggering reconciliation as in the join case.

Data Lifecycle Management. The private data store supports two complementary mechanisms for data lifecycle management, satisfying **R6** (Data Lifecycle Management).

Automatic expiry. Each collection defines a block-to-live (BTL) retention parameter τ. Any private data committed more than τ blocks ago is considered expired. A background process periodically purges expired private data while preserving the on-chain hashes, keeping the historical audit trail tamper-evident.

Explicit purge. For finer-grained control—e.g., to comply with the Right to be Forgotten under the EU GDPR [14] on a per-key basis—the `PurgePrivateData`(c, k) API allows a smart contract to permanently remove a specific key from both the current private state and all historical versions stored in the private data store. This is distinct from `DelPrivateData`, which only removes the key from the current state while retaining historical write-sets. When a purge is committed, the peer records a *purge marker* keyed by $(c, H(k))$ that permanently indicates the key has been purged at a given block and transaction index. A *hashed index* maps each hashed key to the block and transaction locations of its historical

write-sets, enabling efficient background cleanup without scanning the entire ledger state.

Purge interacts with the reconciliation protocol: when a peer responds to a reconciliation or reactive retrieval request, it filters out keys that have been purged locally, returning a partial write-set rather than a complete one. The requesting peer accepts partial write-sets by verifying individual key hashes rather than a single hash over the entire collection write-set. If a requesting peer receives data for a key it has already purged, it discards the data. Conversely, if a peer requests data that the responder has already purged, the responder omits those keys and the requester re-queues the request until it observes the purge transaction in a committed block, at which point it removes the entry from its missing data set. In both mechanisms, on-chain hashes are preserved after private data removal, keeping the audit trail tamper-evident; the implications are discussed in §5.

5 THREAT MODEL AND SECURITY ANALYSIS

Our design assumes a decentralized network of organizations in which some nodes may be faulty or malicious. We characterize the trust assumptions for each entity class and the security implications.

Peer Nodes. For *integrity*, every correct peer independently verifies that received private data matches the on-chain hash before accepting it. This local verification makes hash-data integrity (R2) unconditional from the perspective of a correct peer: no pre-image that does not match the committed hash is ever accepted, regardless of how many peers collude. The endorsement policy serves as the trust assumption for *correct execution*—that the smart contract was honestly evaluated and that the committed hash reflects the true computation result. This assumption is not unique to private data: colluding endorsers that satisfy the policy can equally falsify public state by returning fabricated read-write sets, which the ordering service and non-endorsing peers accept at face value. Private data collections therefore introduce no additional trust assumptions beyond what the base protocol already requires. If all endorsing members of a collection collude, they can commit a fraudulent hash while storing the original private data on their own nodes, bypassing local hash verification. This creates a divergence between the on-chain hash and the private data held by collection members. However, the impact is confined to the colluding peers: any correct peer that later joins the collection will reject the mismatching data during reconciliation, and external auditors can detect the inconsistency by requesting the pre-image and verifying it against the ledger. *Confidentiality* is more nuanced. A malicious peer can leak private data it legitimately holds, or refuse to disseminate data to eligible peers; these are inherent limitations of access-control-based privacy, which prevents unauthorized access but cannot constrain how an authorized party uses data it possesses. Confidentiality is therefore bounded by the weakest collection member. Purging offers no stronger assurance: a malicious member can retain private data locally without honoring purge requests, so purge is best-effort rather than a cryptographic guarantee.

Clients. Clients may be malicious, submitting inconsistent private data to different endorsers, fabricating simulation results without sufficient endorsements, or flooding the system with orphan write-sets. The hash mechanism neutralizes inconsistent submissions: if a client sends different private data to different endorsers, each endorser computes a different hash and includes it in its signed response; because the endorsement policy requires matching responses from multiple endorsers, the divergent hashes render the endorsements incompatible and the transaction is rejected during validation. Fabricated simulations are similarly neutralized—peers perform endorsement policy validation *before* attempting to pull any missing pre-images, discarding invalid transactions without fetching their data. For write-set flooding, where a client simulates many transactions but never submits them, a retention window Δ bounds how long orphan entries persist: the transient store automatically purges entries whose simulation height lags the current ledger height by more than Δ blocks, and receiving peers reject disseminated write-sets outside this window. The same retention window defends against a malicious *peer* flooding fake write-sets to other collection members; such activity is tracked via monitoring metrics that let operators identify and remediate faulty peers.

Other Considerations. Only public and hashed read-write sets are submitted to the ordering service, so ordering nodes never see private data payloads; the ordering layer supports both CFT (Raft) and BFT consensus. We assume a partially synchronous network in which partitions are possible but eventually resolve—a partition may delay private data dissemination, but a peer that cannot obtain private data before commit records it as missing and reconciles once connectivity is restored. Private data must be salted by the client before hashing to prevent adversaries from predicting or confirming the underlying data through brute-force hash comparison. The salt must be provided by the client as part of the transient data rather than generated independently by each peer, because all endorsing peers must produce identical hashes to satisfy the endorsement policy. Regarding data lifecycle, purging removes private data from the off-chain store but the on-chain hash is retained on the immutable ledger; this may not fully satisfy regulations such as GDPR that require complete erasure, and we expect peers to be honest in executing purge operations. Finally, private data collections protect payload content but not transaction *metadata*: the hashed read-write sets recorded on the public ledger reference specific collections, and collection membership policies are visible to all network participants, so any peer can infer which organizations are party to a private transaction even without seeing the underlying data. Mitigating this leakage would require techniques such as zero-knowledge set membership proofs [6] or anonymous credentials [7], but even these would only hide which organizations participate; the hashed keys in the read-write sets would still reveal access patterns—analogous to confidential transactions that hide transfer amounts but leave the transaction graph visible.

6 IMPLEMENTATION

The private data collection mechanism targets any blockchain that follows the simulate-order-validate-commit replication protocol. We implemented it within Hyperledger Fabric [1], comprising approximately 33,000 lines of Go code (production logic and tests), shipped in production since v1.2. Figure 4 shows how new components (gray) extend the peer architecture (white).

Private Data Flow. During simulation, the *endorser* writes private write-sets to the *transient store* (❶), a temporary staging area

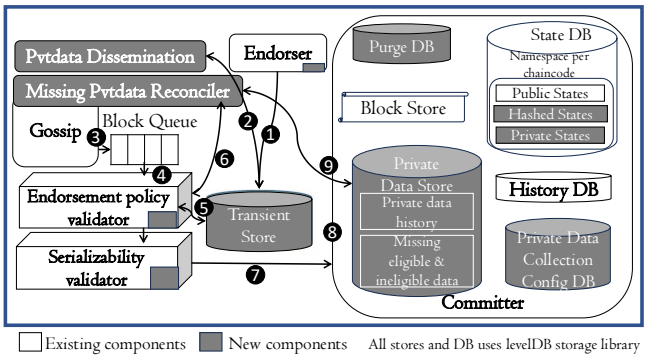

Figure 4: Peer architecture with private data collection components. Pre-existing components (white) form the core transaction pipeline; new components (gray) implement the private data collection mechanism.

Table 4: Workload characteristics and experimental setup.

	Supply Chain	Auction	Trade Finance	EHR
Domain	Farm-to-retail	Sealed-bid auction	Int'l trade	Patient care
Orgs / Peers	10 / 10	6 / 6	12 / 12	8 / 8
PDC collections	17	9	21	13
Only-PDC tx	49.1%	48.1%	45.9%	0.0%
Mixed tx	0.0%	0.0%	0.0%	48.6%
Public-only tx	50.9%	51.9%	54.1%	51.4%
Endorsement policy	OR (any 1)	AND (2/2)	Majority (7/12)	2-of-4 roles
Reads / Writes	1 / 1	2 / 1	1 / 2	3 / 1
VMs†	14	10	16	12

†Each VM: 32 vCPUs (2x8-core Xeon), 64 GB RAM, 12 TB disk, RHEL 9 / Raft orderers. VMs span 4 WAN regions (US-E, US-S, EU, JP). Load is increased until saturation; we report peak tx/s and latency over 300 s.

keyed by transaction ID, while the *Pvtdata Dissemination* component concurrently pushes private write-sets to authorized collection members (❷). When the *gossip* layer delivers an ordered block from the ordering service into the *block queue* (❸), the *endorsement policy validator* checks each transaction against both smart contract-level and collection-level policies (❹). For valid transactions, the peer retrieves private data from its local *transient store* (❺); any data still missing—due to network delays—is pulled from eligible peers by the *Missing Pvtdata Reconciler* (❻). The *serializability validator* then performs blind OCC [26] validation against the public and hashed state partition in the *State DB*, verifying serializability without accessing private data (Theorem 1) (❼). Finally, the *committer* moves validated private data from the transient store to the durable *Private Data Store* (❽), which also tracks eligible and ineligible missing data for future reconciliation, and updates the public, hashed, and private partitions in the *State DB* (❾), where per-smart contract namespace isolation prevents cross-contract access to hashed or private state. A background *Purge DB* process expires private data based on the block-to-live parameter τ, and the `PurgePrivateData` API (Table 2) supports application-controlled deletion by committing purge markers that prevent data from appearing in queries or being reconciled, providing the "right to forget" behavior required by data protection regulations.

Commit-Path Optimizations. Initial microbenchmarks of the commit path revealed a 50% throughput penalty for PDCs (720 vs. 1440 tx/s). Profiling identified cryptographic membership checks and serial disk I/O as the dominant bottlenecks. We applied a series of crash-safe optimizations (Table 3) that reduced the commit-path overhead to just 6%; end-to-end overhead across industry workloads is 3–14% (§7). Key techniques include parallelizing block-store and private-data-store commits, deferring transient-store purge to a background routine, replacing a costly commit-time membership check with an O(1) organization identifier lookup, and adding a transient-store read cache. All optimizations are crash-safe: the block store acts as a write-ahead log (WAL), durably recording each block with its validation results before the State DB or Private Data Store are updated. During recovery, the peer compares the block height in the block store against the heights recorded in the State DB and Private Data Store; any shortfall indicates that the corresponding updates were not applied before the crash, and the peer replays the missing blocks to bring all stores back into sync.

7 EVALUATION

We evaluate four performance criteria: (1) what is the end-to-end overhead of private data relative to all-public transactions? (2) how sensitive is performance to dissemination parameters? (3) how far does the system scale with collection count? and (4) how resilient is the system to membership changes and faults?

Experimental Setup. We design four application workloads modeling multi-party business processes to stress different dimensions of the PDC mechanism (Table 4). Supply Chain maximizes the collection count per organization; Auction isolates dissemination throughput; Trade Finance stresses scalability under strict majority endorsement; and EHR tests read throughput. For each, we compare a *PDC variant* against an *all-public variant* to isolate privacy overhead. In addition, we develop `PDC_BENCH`, a minimal smart contract for controlled micro-benchmarking that exercises all public and private data APIs, enabling direct measurement of individual PDC operations against a public-ledger baseline.

7.1 End-to-End Performance

We compare the PDC and all-public variants of all four workloads at peak throughput, with roughly 40–50% of transactions touching private data collections in each PDC variant. Figure 5 shows throughput and commit latency side by side. Across all four workloads, PDCs consistently reduce peak throughput: 3% for Auction, 7.5% for Supply Chain, 10% for Trade Finance, and 14% for EHR. The overhead correlates with per-transaction complexity. EHR sees the largest drop because it is the most read-heavy workload (3 reads per tx) and each private-data read adds a private-data-store lookup. Trade Finance suffers from its strict 7-of-12 endorsement policy: the additional dissemination and commit-path work compounds an already long critical path. Auction and Supply Chain, with simpler profiles and lighter endorsement policies, see the smallest overheads. Commit latency increases by 4–17% across the workloads,

Table 3: Commit-path optimizations.

Optimization Step	tx/s	Gain
PDC baseline (No optimizations)	720	—
+ O(1) membership identifier check	880	+22%
+ Background transient-store purge	1040	+44%
+ Parallel block and private data store commit	1152	+60%
+ Parallel purge and state DB commit	1248	+73%
+ Transient-store cache	1344	+87%
Public data only (Reference baseline)	1440	—

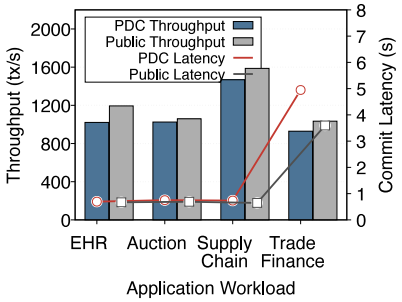

Figure 5: End-to-end throughput and commit latency for PDC and all-public.

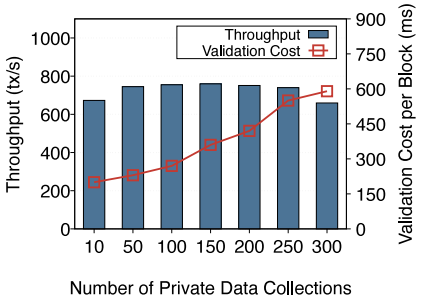

Figure 6: Throughput and per-block validation cost as collections scale to 300.

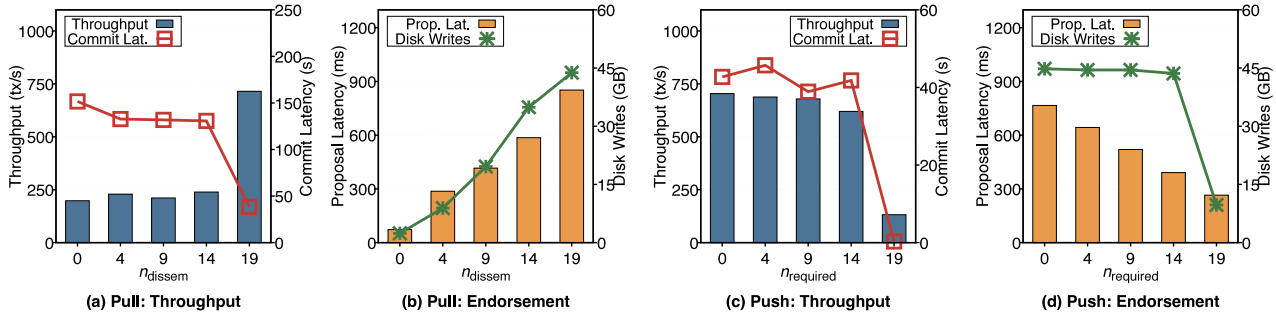

Figure 7: Dissemination parameter sensitivity on a 20-organization WAN network: pull-dominant (a, b) varying n_{dissem} and push-with-quorum (c, d) varying $n_{required}$.

consistent with the additional hash verification and private-data-store operations on the commit path.

Where does the overhead originate from? Per-node Prometheus metrics and disk I/O counters collected during each 300 s run point to three sources. (1) *CPU:* per-transaction cost rises 10–20% due to hash computation, transient-store writes, and private-data-store flushes—none of which exist in the all-public path. (2) *Network:* because the ordering service never sees private payloads, they must be distributed point-to-point via gossip, increasing per-transaction traffic by 10–15%; missed payloads trigger reactive retrieval that further inflates peer-to-peer traffic. (3) *Disk reads:* committers query the transient store and private-data store to match payloads against on-chain hashes, increasing read I/O by 2–4×. Notably, total disk *writes* decrease: in the all-public path every peer writes the full payload, whereas with PDCs the ledger records only a hash and only authorized peers persist the private payload—a storage benefit of selective replication.

7.2 Scalability

To stress-test the per-collection overhead, we deploy a 10-org WAN network and scale the number of private data collections attached to a single smart contract from 10 to 300 in increments of 50 (configurations c10–c300). The workload uses `PDC_BENCH`: each transaction writes to a single randomly selected collection; all other parameters (block size, endorsement policy, value size) remain fixed. Figure 6 plots throughput (bars) and per-block validation cost (line) as the collection count grows. The system sustains peak throughput (740–760 tx/s) up to 250 collections. Starting at c150, average latency begins to stretch; at c300, throughput drops by 11% (to 660 tx/s) while average latency spikes to 24 s. Disk I/O, ledger commit time, and state DB metrics remain flat throughout the range, confirming that the storage layer is *not* the bottleneck. The dominant cost is CPU time spent in the validation phase: per-block validation cost rises linearly from 0.20 s at c10 to 0.59 s at c300 (Figure 6).

This degradation is *not* an architectural limit but an implementation artifact. The root cause is a serialization shortcut: all collection definitions are stored in a single encoded binary format package per smart contract, and the validator fully deserializes this package for every transaction that touches any collection—even though each transaction touches only one. At 300 collections, this deserialization runs hundreds of times per second, exhausting CPU headroom. Caching the parsed package or indexing individual collection policies would eliminate the linear penalty without architectural changes. Real-world deployments today rarely exceed a few dozen collections per smart contract, well within the flat-throughput region.

7.3 Dissemination Sensitivity

To isolate the effect of the dissemination parameters introduced in Algorithm 1, we deploy a 20-organization WAN network where all

peers share a single private data collection. The workload consists entirely of private-data write transactions, each requiring a single endorsement, running at 600 tx/s for 300 s. We study two dissemination strategy families by varying the collection configuration parameters n_{dissem} and $n_{required}$:

- *Pull-dominant:* The endorsing peer proactively pushes to $n_{dissem} \in \{0, 4, 9, 14, 19\}$ peers before ordering; the remaining $20 - n_{dissem}$ peers retrieve private data reactively at commit time ($n_{required} = 0$, so endorsement latency is unaffected by dissemination).
- *Push-with-quorum:* The endorsing peer pushes to all 20 peers ($n_{dissem} = 20$) but blocks the endorsement response until $n_{required} \in \{0, 4, 9, 14, 19\}$ peers acknowledge receipt.

Figure 7 shows performance under the two strategies. Under pull-dominant dissemination, throughput degrades as n_{dissem} decreases (Figure 7(a)): with $n_{dissem} = 0$, only the endorsing peer holds the private data at commit time, so all 19 remaining peers issue pull requests. Because pulls target random peers, most miss the single data holder and time out. Transactions still commit—peers record the missing data for background reconciliation—but the low cumulative disk writes in Figure 7(b) confirm that most peers have not yet persisted the private data within the measurement window.

Under push-with-quorum, all private data reaches every peer (disk writes remain uniformly high in Figure 7(d)), but requiring WAN acknowledgments before returning the endorsement response creates backpressure. Figure 7(c) shows throughput dropping 81% at $n_{required} = 19$ as the endorser waits for the slowest cross-region peer. Proposal latency rises correspondingly (Figure 7(d)), confirming that the bottleneck shifts from commit-time retrieval to endorsement-time blocking. We conclude that asynchronous push ($n_{dissem} = N-1, n_{required} = 0$) is the optimal WAN configuration: it ensures all peers receive private data before commit time (full disk writes across all peers), maximizes throughput, and eliminates commit-time pull storms and deferred reconciliation.

7.4 Dynamic Membership and Fault Tolerance

We design a two-phase experiment on a five-organization Fabric network (one peer per org). In Phase 1, four organizations populate a shared private data collection with 50,000 key-value pairs. In Phase 2, the collection policy is expanded to a fifth organization via a smart contract upgrade; the new peer reconciles the 50,000 historical records while an additional 50,000 transactions execute concurrently. We run this under two conditions: (i) a *baseline* with no faults, and (ii) a *stress* configuration that crashes the newly joined peer mid-reconciliation and an existing data-holder peer, each followed by recovery after a delay. Correctness is verified by fetching all 100,000 key-value pairs from every peer after stabilization.

In the baseline (Figure 8(a)), throughput rises from 396 tx/s to 474 tx/s when the fifth organization joins, and reconciliation of 50,000 historical records completes at $t \approx 229$ s without visible throughput degradation, confirming that background synchronization is isolated from the transaction pipeline. Under stress (Figure 8(b)), crashing the synchronizing peer (`-org5`) halts reconciliation, but it resumes from its last checkpoint upon recovery (`+org5`). Crashing an existing data-holder (`-org2`) drops throughput from ~500 tx/s to the 4-peer baseline of ~400 tx/s. Despite compound

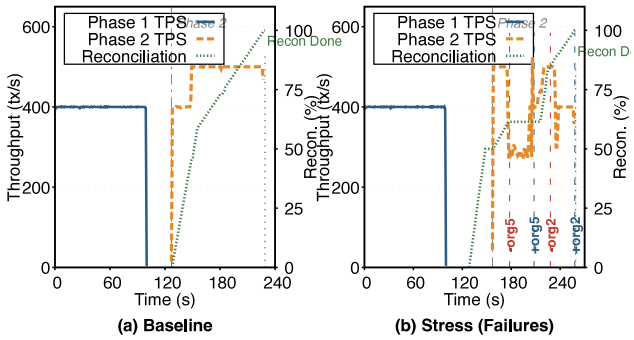

Figure 8: Dynamic membership and fault tolerance. (a) Baseline. (b) Stress with compound peer failures and recoveries.

failures, reconciliation reaches 100% data consistency at $t \approx 259$ s, confirming that the pull mechanism is self-healing.

8 REAL-WORLD IMPACT

Since its introduction in Hyperledger Fabric v1.2, the private data collection mechanism has become the primary privacy primitive in enterprise blockchain deployments spanning food safety, shipping, pharmaceuticals, aerospace, mining, and finance.

Food Safety. Walmart's blockchain-based traceability system, built on the IBM Food Trust network [21, 39], reduced farm-to-store tracing from seven days to 2.2 seconds. It tracks products across produce, meat, dairy, and multi-ingredient categories, with leafy-green suppliers mandated to participate. PDCs protect supplier pricing and process data in bilateral collections while preserving a hash-verified provenance trail for participants and regulators.

Global Shipping. GSBN [24], a consortium of eight major carriers and terminal operators including COSCO SHIPPING Lines, Hapag-Lloyd, and PSA International, has processed over one million shipments across China, the U.S., and Singapore for more than 10,000 customers. PDCs keep Bills of Lading, letters of credit, and customs declarations confidential between parties while the shared ledger reduces cargo-release times from days to hours.

Pharmaceuticals. The MediLedger project [35, 38] unites 24 companies, including Pfizer, Gilead, McKesson, and Walgreens, to track prescription drugs under the U.S. Drug Supply Chain Security Act [37], an approach validated by an FDA pilot. It layers zero-knowledge proofs on PDCs: bilateral terms and verification responses stay private, while ZKPs confirm authenticity without revealing proprietary pricing.

Mining, Aerospace, and Finance. In the $1.8 trillion metals and mining supply chain, MineHub [23] uses PDCs for bilateral contract data across miners, trading houses, financiers, and logistics providers, replacing channel-based sharing that could not scale to dynamic relationships. Honeywell's GoDirect Trade marketplace [20] surpassed $1 million in sales within ten weeks while using PDCs to isolate buyer/seller transaction data and record part provenance to combat counterfeits. Oracle's Blockchain Platform [31]

combines PDCs with Pedersen commitments for wholesale CBDC: bank account numbers stay private while ledger hashes enable auditor verification.

Across deployments, PDCs partition sensitive bilateral data—pricing, contracts, account identifiers, trade documents—while preserving a shared, hash-verified audit trail. The diversity of industries and trust models confirms that the abstraction generalizes beyond any domain.

9 LESSONS FROM PRODUCTION

Membership is never static. Static collection membership proved impractical as consortia routinely added and removed organizations. Membership changes create a cascade of secondary requirements: immutable block-to-live policies, historical data transfer to new members, dual bookkeeping of *eligible*/*ineligible* missing data, atomic reclassification, and background reconciliation. To reduce operational burden, we introduced *implicit collections*—dynamic private namespaces that eliminate explicit definitions for every bilateral relationship.

Replication-layer privacy is necessary but not sufficient. Preventing non-members from *receiving* private data does not prevent smart-contract invocation on an authorized peer; declarative `MemberOnlyRead/Write` flags close this gap at the handler layer. Conversely, non-members often need to verify private data without seeing it, making `GetPrivateDataHash` a first-class interface rather than merely an integrity primitive. Extending state-based endorsement to private data also lets validators enforce ownership rules via hashed keys.

Delete and purge are fundamentally different. Block-number-based expiry (BTL) supports general retention, but GDPR-style erasure requires per-key purge. Removing only current state (`DelPrivateData`) is insufficient if historical write-sets survive; `PurgePrivateData` removes both current state and historical versions. Purge markers filter reads, prevent reconciliation from re-fetching purged data, and guide background cleanup.

Defaults must reflect WAN realities. A 1,000-block transient-store retention window caused premature data loss on slow-consensus networks, forcing a 20× increase. Synchronous dissemination ($n_{required} > 0$) also created endorsement backpressure. Production favored asynchronous push ($n_{required} = 0$), aggressive gossip ($n_{dissem} > 0$), and background reconciliation.

10 RELATED WORK

Channels and Sub-Networks. Hyperledger Fabric's *channels* [1] isolate data via independent ledgers and consensus instances. They are effective but coarse-grained: each sharing relationship may require a separate channel. PDCs provide fine-grained privacy within one channel.

Private Transactions via Off-Chain Managers. Quorum [28] and Hyperledger Besu [22] use external transaction managers for hash-on-chain / data-off-chain privacy, but assume static privacy groups, online participants, and limited public/private state interoperability. PDCs support dynamic membership, offline peers through gossip, and atomic cross-state reads and writes. R3 Corda [3] shares transactions only need-to-know, but input verification requires backchain resolution [3], which can expose historical data and forfeits network-wide queries plus the hash-based audit trail.

Cryptographic and Hardware Privacy. Zerocash [2], Hawk [25], and Zether [4] provide strong cryptographic privacy, but proof generation remains too expensive for high-throughput enterprise workloads. MediLedger [35, 38] shows these techniques can complement PDCs. TEEs (Ekiden [8], CCF [34]) avoid proof cost but introduce hardware dependencies and side-channel risk; MPC (Enigma [40]) and fully homomorphic encryption [18] remain too costly. PDCs instead let authorized peers process plaintext and store only hashes.

Off-Chain State and Database Privacy. State channels and rollups [10, 11, 32] anchor off-chain state on a base ledger, but target permissionless scalability rather than selective access control in permissioned networks. CryptDB [33] and differential privacy [9] protect database queries, whereas PDCs control replication-layer access.

Positioning. PDCs achieve fine-grained privacy through selective replication, on-chain hashes, and blind validation rather than expensive cryptography or trusted hardware. Alternatives sacrifice at least one required property: channels forfeit granularity, Corda-style designs forfeit the global audit trail, proofs forfeit throughput, and enclaves forfeit deployment flexibility.

11 CONCLUSION

We presented *private data collections* (PDCs): private data stays on authorized peers while hashes are recorded on the replicated ledger. On-chain hashes form a *conflict-complete proxy* for private state (Theorem 1), so OCC [26] validation detects the same conflicts as private-state checks without exposing data. A transient store, hashed read-write sets, and dissemination protocol deliver privacy, atomicity, integrity, and auditability without specialized hardware, expensive cryptography, or channel overhead. The Hyperledger Fabric implementation, production-ready since v1.2, adds 3–14% overhead and has seen cross-industry deployment.

REFERENCES

[1] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. 2018. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In *Proceedings of the Thirteenth EuroSys Conference* (Porto, Portugal) *(EuroSys '18)*. ACM, New York, NY, USA, Article 30, 15 pages. https://doi.org/10.1145/3190508.3190538 Accessed: 2026-07-09.

[2] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy (SP)*. IEEE, 459–474. https://doi.org/10.1109/SP.2014.36

[3] Richard Gendal Brown, James Carlyle, Ian Grigg, and Mike Hearn. 2016. Corda: An Introduction. R3 Technical White Paper. https://docs.r3.com/en/pdf/corda-technical-whitepaper.pdf Accessed: 2026-07-09.

[4] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. 2019. Zether: Towards Privacy in a Smart Contract World. *IACR Cryptology ePrint Archive* 2019 (2019), 191.

[5] Vitalik Buterin. 2014. Ethereum: A next-generation smart contract and decentralized application platform. https://ethereum.org/en/whitepaper/ Accessed: July 31, 2019.

[6] Jan Camenisch, Rafik Chaabouni, and abhi shelat. 2008. Efficient Protocols for Set Membership and Range Proofs. In *Advances in Cryptology — ASIACRYPT 2008*. Springer, 234–252. https://doi.org/10.1007/978-3-540-89255-7_15

[7] Jan Camenisch and Anna Lysyanskaya. 2001. An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation. In

Advances in Cryptology — EUROCRYPT 2001. Springer, 93–118. https://doi.org/10.1007/3-540-44987-6_7

[8] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song. 2019. Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contracts. In *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P '19)*. IEEE, 185–200.

[9] Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Theoretical Computer Science* 9, 3–4 (2014), 211–407. https://doi.org/10.1561/0400000042

[10] Ethereum Foundation. 2023. Optimistic Rollups. https://ethereum.org/en/developers/docs/scaling/optimistic-rollups/. Accessed: 2026-07-09.

[11] Ethereum Foundation. 2023. ZK-Rollups. https://ethereum.org/en/developers/docs/scaling/zk-rollups/. Accessed: 2026-07-09.

[12] European Commission. 2023. Food Safety Policy. https://food.ec.europa.eu/horizontal-topics/general-food-law_en. Accessed: 2026-07-09.

[13] European Parliament and Council of the European Union. 2002. Regulation (EC) No 178/2002 – General Food Law. https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=celex:32002R0178. Accessed: 2026-07-09.

[14] European Parliament and Council of the European Union. 2016. General Data Protection Regulation (GDPR) – Regulation (EU) 2016/679. https://gdpr-info.eu/ Accessed: 2026-07-09.

[15] Food and Agriculture Organization of the United Nations. 2017. Food Traceability Guidance. http://www.fao.org/3/a-i7665e.pdf. Accessed: 2026-07-09.

[16] Food Standards Australia New Zealand. 2023. Food Standards Australia New Zealand. https://www.foodstandards.gov.au/. Accessed: 2026-07-09.

[17] Jai Singh Arun; Genarro Cuomo; Nitin Gaur. 2019. *Blockchain for Business* (paperback ed.). Addison-Wesley Professional. https://www.pearson.com/en-us/subject-catalog/p/blockchain-for-business/P200000009423/9780135581353 Accessed: 2026-07-09.

[18] Craig Gentry. 2009. Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing (STOC '09)*. ACM, New York, NY, USA, 169–178. https://doi.org/10.1145/1536414.1536440

[19] HHS. [n.d.]. HIPAA Privacy Rule. https://www.hhs.gov/hipaa/for-professionals/privacy/index.html Accessed: 2026-07-09.

[20] Hyperledger Foundation. 2019. Case Study: Honeywell Aerospace Creates Online Parts Marketplace with Hyperledger Fabric. https://www.lfdecentralizedtrust.org/case-studies/honeywell-case-study. Accessed: 2026-07-09.

[21] Hyperledger Foundation. 2019. How Walmart Brought Unprecedented Transparency to the Food Supply Chain with Hyperledger Fabric. https://web.archive.org/web/20210308235803/https://www.hyperledger.org/wp-content/uploads/2019/02/Hyperledger_CaseStudy_Walmart_Printable_V4.pdf. Accessed: 2026-07-09.

[22] Hyperledger Foundation. 2020. Hyperledger Besu. https://www.lfdecentralizedtrust.org/projects/besu. Accessed: 2026-07-09.

[23] Hyperledger Foundation. 2021. Case Study: Managing the Metal and Mining Industry's Supply Chain. https://www.lfdecentralizedtrust.org/case-studies/krypc-minehub-case-study. Accessed: 2026-07-09.

[24] Hyperledger Foundation. 2022. GSBN Simplifies Global Trade with Hyperledger Fabric. https://www.lfdecentralizedtrust.org/case-studies/gsbn-case-study. Accessed: 2026-07-09.

[25] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. 2016. Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts. In *2016 IEEE Symposium on Security and Privacy (SP)*. 839–858.

[26] H. T. Kung and John T. Robinson. 1981. On Optimistic Methods for Concurrency Control. *ACM Trans. Database Syst.* 6, 2 (June 1981), 213–226. https://doi.org/10.1145/319566.319567 Accessed: 2026-07-09.

[27] Hagar Meir, Artem Barger, and Yacov Manevich. 2019. Increasing Concurrency in Hyperledger Fabric. In *Proceedings of the 12th ACM International Conference on Systems and Storage* (Haifa, Israel) *(SYSTOR '19)*. ACM, New York, NY, USA, 179–179. https://doi.org/10.1145/3319647.3325841 Accessed: 2026-07-09.

[28] JP Morgan-Chase. [n.d.]. Quorum Private Transactions. https://github.com/jpmorganchase/quorum Accessed: 2026-07-09.

[29] Senthil Nathan, Chander Govindarajan, Adarsh Saraf, Manish Sethi, and Praveen Jayachandran. 2019. Blockchain Meets Database: Design and Implementation of a Blockchain Relational Database. *Proc. VLDB Endow.* 12, 11 (July 2019), 1539–1552. https://doi.org/10.14778/3342263.3342632 Accessed: 2026-07-09.

[30] National Center for Biotechnology Information. 2023. Escherichia Coli. https://www.ncbi.nlm.nih.gov/books/NBK507845/. Accessed: 2026-07-09.

[31] Oracle. 2024. Robust, Scalable and Efficient System for Confidential Token Transactions: Oracle's Breakthrough on Hyperledger Fabric. https://docs.oracle.com/en/cloud/paas/blockchain-cloud/digitalassetsoci/confidential-transactions.html. Accessed: 2026-07-09.

[32] Joseph Poon and Thaddeus Dryja. 2016. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. https://lightning.network/lightning-network-paper.pdf. Accessed: 2026-07-09.

[33] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2011. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (SOSP '11)*. ACM, New York, NY, USA, 85–100. https://doi.org/10.1145/2043556.2043566

[34] Mark Russinovich, Edward Ashton, Christine Avanessians, Miguel Castro, Amaury Chamayou, Sylvan Clebsch, Manuel Costa, Cédric Fournet, Matthew Kerner, Sid Krishna, Julien Maffre, Thomas Shermer, and Olya Roy. 2019. *CCF: A Framework for Building Confidential Verifiable Replicated Services*. Technical Report MSR-TR-2019-16. Microsoft Research. https://github.com/microsoft/CCF Accessed: 2026-07-09.

[35] Ben Taylor. 2019. Hyperledger for Healthcare: How Fabric Drives the Next-Generation Pharma Supply Chain. https://www.lfdecentralizedtrust.org/blog/2019/12/02/hyperledger-for-healthcare-how-fabric-drives-the-next-generation-pharma-supply-chain. Accessed: 2026-07-09.

[36] Parth Thakkar and Senthilnathan Natarajan. 2021. Scaling Blockchains Using Pipelined Execution and Sparse Peers. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC '21)*. ACM, New York, NY, USA, 392–407. https://doi.org/10.1145/3472883.3486975

[37] U.S. Congress. 2013. Drug Supply Chain Security Act (DSCSA), Title II of the Drug Quality and Security Act. https://www.fda.gov/media/168283/download. Accessed: 2026-07-09.

[38] U.S. Food and Drug Administration. 2023. DSCSA Pilot Project Program: MediLedger Final Report. https://www.fda.gov/media/168283/download. Accessed: 2026-07-09.

[39] Walmart Inc. 2018. Walmart and Sam's Club to Require Real-Time, End-to-End Food Traceability with Blockchain. https://corporate.walmart.com/newsroom/2018/09/24/in-wake-of-romaine-e-coli-scare-walmart-deploys-blockchain-to-track-leafy-greens. Accessed: 2026-07-09.

[40] Guy Zyskind, Oz Nathan, and Alex Pentland. 2015. Enigma: Decentralized Computation Platform with Guaranteed Privacy. arXiv:1506.03471. https://arxiv.org/abs/1506.03471 Accessed: 2026-07-09.