Fold DB: Compute Without Exposure

Tom Tang

March 4, 2026

Abstract

Fold DB is a database in which data is never accessed directly. Users interact with *folds*—named interfaces that sit in front of stored values and enforce access policies on every operation. Each fold defines which fields are visible, what transformations are applied, and who may read or write, using four conjunctive mechanisms: owner-assigned trust distance, cryptographic capabilities with bounded quotas, payment gates, and lattice-based security labels. Folds compose: one fold can derive fields from another, forming a directed acyclic graph in which each step applies its own independent policy. Lattice-based security labels prevent information from flowing downward through these compositions. Every access is authenticated and recorded in an append-only audit log. We formalize this model and prove a data-minimality property: a query can never return more information than the caller is authorized to see.

1 Introduction

Sharing data across trust boundaries is a fundamental problem in multi-tenant and cloud systems. Traditional access control operates at the perimeter—once data reaches a processing host, the system has limited ability to constrain what happens to it. Fine-grained, per-field enforcement is rare. Cryptographic proof that a query revealed only what was authorized is rarer still. Economic controls (who paid for access) are typically handled outside the data layer entirely.

Fold DB addresses these gaps with a single abstraction: the *fold*. A fold is a policy-enforcing interface over stored data. Raw values are never returned to callers. Instead, each query passes through a fold that checks the caller's trust distance, verifies cryptographic credentials and payment, applies any registered transformations, and returns only the authorized projection. If any check fails, the query returns nothing—no partial results, no error messages that leak structure. The fold is the only way to reach the data.

This paper makes three contributions:

1. A formal model of folds as composable, policy-enforcing computation interfaces, with a monadic semantics that guarantees clean failure propagation.

2. A layered access-control model combining trust distance, cryptographic capabilities, lattice-based information flow, and economic gates.

3. A data-minimality property with proof: queries never return more than the authorized projection.

2 Overview by Example

Consider a hospital where a patient stores a medical record containing three fields: `name`, `diagnosis`, and `lab_results`. The patient (the data owner, trust distance $\tau = 0$) creates three folds over the same underlying data.

Fold 1: Clinical access (F_{clin}). Exposes all three fields unchanged, with policy $W_1 R_1$—writable and readable by anyone with trust distance ≤ 1. This means the patient's doctor ($\tau = 1$) can both read the record and write updates (e.g., adding a new diagnosis), not just the patient. Every write is cryptographically signed, and the append-only store retains all previous versions.

Fold 2: Research access (F_{res}). Exposes the same three fields, but with an irreversible hash transform on `name` (replacing it with a de-identified token). Policy is $W_0 R_3$—readable at greater distance, but only the owner can write (researchers should not be modifying de-identified records).

Fold 3: Composite analytics (F_{agg}). Derives a new field `risk_score` by applying a scoring transform T_{score} to the `diagnosis` and `lab_results` fields of F_{res}. Policy is $W_0 R_5$. Because F_{agg} depends on F_{res}, the researcher's access context must satisfy *both* folds' policies to reach the score.

Now three users query the system:

- **Attending physician** ($\tau = 1$). Queries F_{clin}. Trust distance is within bounds ($1 \leq 1$), so she receives all three fields in cleartext. The query is logged.
- **External researcher** ($\tau = 3$). Queries F_{res}. Trust distance is within bounds ($3 \leq 3$), so he receives the hashed name, diagnosis, and lab results. He cannot recover the patient's identity because the hash is irreversible. If he queries F_{clin} instead, it returns nothing ($3 > 1$). He can also query F_{agg} ($3 \leq 5$) to get the risk score—but only because F_{res} also passes at $\tau = 3$.
- **Unauthorized user** ($\tau = 10$). Every fold returns nothing. The attempt is logged.

This example illustrates the core properties: the same data supports multiple access patterns through different folds; irreversible transforms prevent information recovery; folds compose into a DAG with independent policy checks at each node; and failure at any level produces no output.

3 The Fold Model

3.1 Definitions

A *fold* F is an interface over a set of fields $\{f_1, \ldots, f_n\}$. Each field f_i carries:

- A value V_i (the stored or derived datum).
- A security label ℓ_i from a lattice $(\mathcal{L}, \sqsubseteq)$.
- A trust-distance policy $W_{n_i} R_{m_i}$.
- Optional capability constraints $\text{WX}_k(pk)$ or $\text{RX}_k(pk)$.

A *query* evaluates a fold under an access context $\mathcal{C} = (u, \tau, K)$, where u identifies the caller, τ is the caller's trust distance, and K is the set of public keys the caller holds. Payment status is resolved per-fold at evaluation time via the predicate $\mathcal{P}(u, F)$ (Section 4), rather than being carried in the context, since the same caller may have paid for some folds but not others. The result is either the authorized projection π or nothing:

$$F(\mathcal{C}) = \begin{cases} \text{Just}(\pi) & \text{if all policies and payments are satisfied} \\ \text{Nothing} & \text{otherwise} \end{cases}$$

3.2 Monadic Semantics

The type of a fold computation is:

$$\text{Fold}[a] = \mathcal{C} \to \text{Maybe } a$$

This is the composition of the reader monad (threading $\mathcal{C}$) and the maybe monad (representing policy failure). The operations are:

$$\text{return}(x) = \lambda\mathcal{C}.\ \text{Just}(x) \tag{1}$$

$$m \gg\!= f = \lambda\mathcal{C}.\ \begin{cases} f(a)(\mathcal{C}) & \text{if } m(\mathcal{C}) = \text{Just}(a) \\ \text{Nothing} & \text{if } m(\mathcal{C}) = \text{Nothing} \end{cases} \tag{2}$$

These inherit the monad laws (left identity, right identity, associativity) from the standard reader-maybe composition [6]. The key consequence is that failure propagates: if any step in a composed fold chain yields Nothing, the entire chain yields Nothing without evaluating subsequent data steps. (The audit service still records the failed attempt—short-circuiting applies to data evaluation, not logging.)

3.3 Transforms and Field Derivation

A *transform* T derives a field in one fold from a field in another. Each field has at most one active transform. At query time, a transformed field f_i with transform T and source fold S resolves to $f_i(\mathcal{C}) = T(S(\mathcal{C}))$—the transform is applied to the source fold's output under the same access context. Fields without transforms hold their values independently.

Transforms are classified by reversibility:

- **Reversible**: the field is readable and writable. Writes apply T^{-1} and propagate to the source fold. In chains ($T_1 \circ T_2 \circ \cdots$), inverses compose back to the origin. Concurrent writes to the same upstream field use last-write-wins for the *active* value (the value returned by subsequent reads), but all prior values remain in the append-only store and are available for audit and rollback.
- **Irreversible**: the field is read-only. Writes are rejected. The original value cannot be recovered from the output (e.g., cryptographic hashing).

3.4 Fold Composition

Folds form a directed acyclic graph (DAG) through their transform dependencies. A fold F_k that derives fields from F_{k-1} evaluates as:

$$F_k = \lambda\mathcal{C}.\ F_{k-1}(\mathcal{C}) \gg\!= T_k$$

Each fold in the chain applies its own policies independently. This enables:

- Modular reuse of validated sub-computations.
- Policy-preserving aggregation of multi-source data.
- Fault isolation—failure of one fold produces Nothing, not a partial result.

Cycles are rejected at fold registration time, guaranteeing termination.

4 Access Control

Access to a field requires passing four independent checks. All are conjunctive: every applicable check must succeed.

4.1 Trust Distance

Definition 1 (Trust Distance)**.** *The trust distance $\tau(u, o) \in \mathbb{N}_0$ between user u and data owner o is a non-negative integer. $\tau = 0$ denotes the owner. Trust is* additive*: if the owner assigns $\tau(a, o) = n$ and user a assigns $\tau(b, a) = m$, then $\tau(b, o) = n + m$ by default. This allows trust to propagate through chains without requiring the owner to assign a distance to every user individually. When multiple paths exist between a user and the owner, the system uses the* shortest path*—i.e., $\tau(u, o) = \min_{paths}(\text{sum of distances along path})$. The owner may override any derived distance with an explicit assignment—for example, setting $\tau(b, o) = 5$ even though the shortest additive chain yields 2. Explicit assignments take precedence over all derived distances. Trust distances are mutable and take effect on the next query.*

Each field carries a trust-distance policy $W_n R_m$:

$$\text{Writable if } \tau \leq n, \qquad \text{Readable if } \tau \leq m$$

Any user whose trust distance satisfies $\tau \leq n$ may write to the field—not only the owner. Every write, regardless of author, must carry a cryptographic signature ($\sigma = \text{Sign}(sk, D)$) that authenticates the writer. The append-only store records every version; previous values are never deleted, enabling full history traversal, audit, and rollback.

Revocation. When the owner changes a trust distance assignment (e.g., revoking Alice from $\tau = 1$ to $\tau = 10$), the change takes effect on the next query. In-flight queries that began before the change complete under their original context; no results are retroactively withdrawn. Transitive dependents are updated automatically: if Alice's trust distance increases, every user whose derived distance flows through Alice is recomputed on their next query. There is no caching of trust distances—each query resolves the current distance at evaluation time.

4.2 Cryptographic Capabilities

Fields may additionally require key-based authorization with bounded quotas:

- $\text{WX}_k(pk)$: grants write access to the holder of public key pk. The counter k decrements with each write; access is revoked when $k = 0$.
- $\text{RX}_k(pk)$: grants read access to the holder of pk. The counter k decrements with each read, enforcing bounded disclosure.

When both trust-distance and capability constraints are present, the caller must satisfy both. Neither overrides the other.

The data owner manages capability lifecycle: they can issue a fresh capability $\mathrm{RX}_{k'}(pk)$ to the same key at any time, resetting the quota. Quotas cannot be increased in place—the owner revokes the existing capability and issues a new one. All capability grants and revocations are recorded in the audit service.

4.3 Security Labels

Each field carries a label ℓ from a lattice $(\mathcal{L}, \sqsubseteq)$, following Denning's model [3]. A transform producing field f_j from f_i is permitted only if $\ell_i \sqsubseteq \ell_j$: information flows to equal or higher security levels, never downward. This constraint is enforced twice: statically at transform registration (the declared labels must satisfy the ordering) and at query time (the system verifies that the runtime label assignments still satisfy $\ell_i \sqsubseteq \ell_j$, since labels may be changed after registration). This prevents a fold from laundering sensitive data into a lower-classified output.

4.4 Payment Gates

A fold can require payment as a condition of access. The cost is a function $C : \mathbb{N}_0 \to \mathbb{R}_{\geq 0}$ of trust distance, defined by the fold owner. Any monotonically non-decreasing function is valid—common choices include linear ($C(\tau) = a + b\,\tau$) and exponential ($C(\tau) = a\,e^{b\,\tau}$), but the system imposes no restriction beyond non-decreasing monotonicity (closer users pay no more than distant ones). The payment predicate $\mathcal{P}(u, F)$ holds when user u has paid $C(\tau(u, o))$ for fold F. Queries return Nothing unless $\mathcal{P}$ holds, integrating economic controls into the same enforcement path as trust distance and capabilities.

5 Security Analysis

5.1 Threat Model

Fold DB addresses four threats:

- **Unauthorized access.** Every query passes through a fold that checks trust distance, capabilities, and payment before returning any data. Failure yields Nothing—no data, no structural leakage—and the attempt is logged.
- **Data tampering.** Every write—by the owner or any authorized user—requires a cryptographic signature ($\sigma = \mathrm{Sign}(sk, D)$) binding the writer's identity to the data. The append-only store is immutable: entries are never modified or deleted after insertion. All previous versions are retained, providing a complete, auditable history. Inconsistencies between a signed write and the stored record are detectable.
- **Information leakage.** Three mechanisms work in concert: the monadic bind prevents partial results on policy failure; irreversible transforms prevent inversion of derived outputs; and lattice-based labels prevent transforms from downgrading data to a lower classification. The data-minimality property (Property 1) formalizes this guarantee.
- **Repudiation.** Every access, payment, and transformation is recorded in the append-only audit log with user identity, timestamp, fold, and operation. Signed entries are non-repudiable.

5.2 Failure Semantics

On any policy or payment failure, the fold returns Nothing and logs the event. In a composed chain where F_k depends on F_{k-1}: if $F_{k-1}(\mathcal{C}) = \text{Nothing}$, the bind short-circuits and $F_k(\mathcal{C}) = \text{Nothing}$ without evaluating T_k. No intermediate values are exposed.

5.3 Encryption at Rest

All data at rest—fold definitions, stored values, and audit logs—is encrypted using authenticated encryption (AEAD) with per-dataset keys derived from a master secret via HKDF. The append-only store logs key usage to support revocation and rotation.

5.4 Data Minimality

Property 1 (Data Minimality)**.** *For any query evaluated under access context $\mathcal{C}$:*

$$F(\mathcal{C}) = \text{Just}(\pi) \implies \pi \subseteq \text{Auth}(\mathcal{C}, F)$$

where $\text{Auth}(\mathcal{C}, F)$ *is the set of field values for which (i) $\tau \leq m$ for the field's $W_n R_m$ policy, (ii) all capability quotas are positive, (iii) $\mathcal{P}(u, F)$ holds, and (iv) security labels permit the flow.*

Proof sketch. By structural induction on the fold DAG. Note that fold evaluation is all-or-nothing: if any field in a fold fails any of the four checks (trust distance, capability quota, payment, security label), the entire fold returns Nothing.

Base case. A fold with no transforms returns stored values directly. The evaluation checks every field's trust-distance policy ($\tau \leq m$), capability constraints ($pk \in K$ with positive quota), payment predicate ($\mathcal{P}(u, F)$), and security labels against $\mathcal{C}$. If all four conditions hold for every field, the fold returns $\text{Just}(\pi)$ where $\pi \subseteq \text{Auth}(\mathcal{C}, F)$. If any condition fails on any field, the fold returns Nothing.

Inductive step. Suppose fold F_k derives fields from F_{k-1} via transform T_k, and F_{k-1} satisfies data minimality by hypothesis. Then either $F_{k-1}(\mathcal{C}) = \text{Nothing}$ (in which case $F_k(\mathcal{C}) = \text{Nothing}$ by monadic bind) or F_k receives $\pi_{k-1} \subseteq \text{Auth}(\mathcal{C}, F_{k-1})$. The transform T_k produces derived values from its inputs—it cannot inject new data, and the label constraint $\ell_i \sqsubseteq \ell_j$ prevents downward flow. The evaluation of F_k then applies all four checks against F_k's own policies, yielding either $\pi_k \subseteq \text{Auth}(\mathcal{C}, F_k)$ or Nothing.

Thus data minimality is preserved at each composition step. □

6 System Architecture

A Fold DB node consists of five components:

- **Registry**: stores fold definitions, field metadata, and policy configurations.
- **Execution Engine**: evaluates fold computations under a given access context.
- **Transform Library**: a set of vetted, registered transformation functions.
- **Append-Only Store**: immutable log of all data writes. Every value ever written is retained here, enabling history traversal and rollback. This component handles the data itself.

- **Audit Service**: records every access event—reads, failed queries, payment transactions, and trust distance changes. This component handles metadata about operations on the data, not the data itself.

The model presented in this paper addresses single-node operation. All trust assignments, policies, and audit logs are managed by one node. Multi-node federation—where queries span nodes with independent trust domains—is discussed in Section 10.

7 Universal Fold Service

The fold model separates *what computation runs* from *whose data it runs on*. A Universal Fold Service (UFS) exploits this separation: it publishes a shared fold schema—a fixed set of field names, types, transforms, and policies—that any data owner can adopt. Code written against a published schema is deterministic and portable: it runs identically across all users who have instantiated that schema, regardless of the underlying data.

7.1 Schema Publication and Adoption

A UFS schema Σ specifies:

- A set of field names and types $\{(f_1, t_1), \ldots, (f_n, t_n)\}$.
- A set of transforms $\{T_1, \ldots, T_k\}$ with their reversibility classifications.
- Default access policies $\{W_{n_i} R_{m_i}\}$. When an owner adopts the schema, the system enforces that their local fold's write and read thresholds do not exceed the schema defaults: $n_i^{\text{local}} \leq n_i^{\Sigma}$ and $m_i^{\text{local}} \leq m_i^{\Sigma}$. Owners may tighten (lower the thresholds) but not loosen them. A fold that violates these bounds is rejected at registration.
- Security label constraints.

Any data owner can *adopt* Σ by creating a local fold that conforms to it, binding their own data to the schema's fields. The UFS registry records which users have adopted which schemas, enabling discovery without exposing data.

7.2 Deterministic Cross-User Computation

A program P written against schema Σ is a function from Σ-typed fold outputs to a result. Because the schema fixes the field names, types, and transform semantics, P is deterministic: given the same fold output, it produces the same result, regardless of which user's data produced that output. The UFS can execute P across any subset of adopters:

$$\text{UFS}(P, \Sigma, U) = \{P(F_u^{\Sigma}(\mathcal{C}_u)) : u \in U,\ F_u^{\Sigma}(\mathcal{C}_u) \neq \text{Nothing}\}$$

where U is the set of target users, F_u^{Σ} is user u's fold conforming to Σ, and $\mathcal{C}_u$ is the access context for that user. The result is a set of program outputs with no user identifiers attached—the caller cannot determine which user produced which result. Users whose policies reject the query are silently excluded—the program never sees their data or learns of their existence.

7.3 Examples

Population health screening. A public health agency publishes schema Σ_{health} with fields `age`, `bmi`, `blood_pressure`, and `cholesterol`. Patients across many hospitals adopt the schema through their own folds, each with their own trust-distance policies. The agency writes a risk-stratification program P_{risk} against Σ_{health}. The UFS runs P_{risk} across all adopters: each patient's fold independently checks whether the agency's trust distance permits access. Patients who have granted access contribute a risk score; others are silently excluded. The agency receives a set of scores without ever seeing raw medical records, and without knowing which patients declined.

Cross-institution credit scoring. A regulator publishes Σ_{credit} with fields `income`, `debt_ratio`, `payment_history`, and `account_age`. Banks and fintech companies adopt the schema, each mapping their internal data to the standard fields. A scoring model P_{score} runs identically across all adopters. Because the schema fixes the field semantics and types, the model produces comparable scores regardless of the institution's internal data format. Each institution's fold enforces its own policies: some may require payment, others may restrict access to regulators at $\tau \leq 2$. The model code is written once and never modified per institution.

Federated machine learning. A research consortium publishes Σ_{training} with feature fields and a label field. Each member institution adopts the schema with an irreversible differential-privacy transform on sensitive features. A training program P_{train} queries each member's fold and receives only the transformed, privacy-preserving feature values—never the raw data. The UFS ensures that P_{train} sees only the outputs permitted by each institution's policies. Institutions that revoke access mid-training are silently dropped—the program receives Nothing for subsequent queries and continues with the remaining participants.

7.4 Properties

The UFS inherits all guarantees from the fold model:

- **Data minimality**: each user's fold returns only their authorized projection; the program cannot access more.
- **Policy independence**: the schema defines maximum access bounds; each adopter controls policy within those bounds by setting their own trust distances, capabilities, and payment requirements. Owners can tighten access but not loosen it beyond the schema's defaults.
- **Determinism**: the program's behavior depends only on the schema and the fold outputs, not on which user's data is being processed.
- **Graceful exclusion**: users who deny access are invisible to the program—no error, no enumeration, no side channel.

8 Universal Transform Registry

Transforms are bound to folds: a transform exists only as part of a fold definition, and its execution is subject to the fold's access policies. The *Universal Transform Registry* (UTR) extends this principle to a shared catalog. It is the single source of truth for transform definitions across all Fold DB nodes, ensuring that a transform used in one fold behaves identically when adopted by another.

8.1 Registration and Validation

A transform T submitted to the UTR must declare:

- **Input/output types**: the field types it consumes and produces.
- **Reversibility**: whether an inverse T^{-1} is provided.
- **Label constraint**: the minimum output label ℓ_{out} such that $\ell_{\text{in}} \sqsubseteq \ell_{\text{out}}$.
- **Determinism attestation**: a proof or test suite demonstrating that T is a pure function—same input always yields same output, with no side effects.

Because transforms are bound to folds, registration in the UTR automatically validates the transform against the fold model's constraints. A transform that violates label ordering, claims reversibility without a valid inverse, or exhibits non-determinism is rejected. Accepted transforms receive a content-addressed identifier (a hash of the transform definition), making them immutable and globally referenceable.

8.2 Reuse Across Schemas

Once registered, a transform can be referenced by any UFS schema. This creates a shared library of vetted building blocks:

- A schema author selects transforms from the UTR by identifier rather than reimplementing them.
- All folds using the same transform identifier are guaranteed identical behavior.
- Updates require registering a new transform (new hash); existing references are unaffected.

8.3 Examples

Standardized anonymization. A healthcare standards body registers $T_{\text{k-anon}}$, a k-anonymization transform that generalizes quasi-identifiers (e.g., replacing exact age with a 5-year range). The transform is marked irreversible. The output retains the input's security label (ℓ_{PII}) because the lattice model tracks provenance, not information content—even though the output is de-identified, it was derived from PII and carries that label. A downstream fold can assign a lower label only if an explicit declassification policy permits it. Any schema that needs de-identification—patient records, insurance claims, clinical trials—references the same $T_{\text{k-anon}}$ by hash. Every adopter gets identical anonymization semantics without reimplementing or re-auditing the logic.

Currency conversion. A financial data consortium registers T_{fx}, a reversible transform that converts monetary values between currencies using a reference exchange rate. The inverse T_{fx}^{-1} converts back. Because it is registered as reversible, folds using T_{fx} allow writes: a user can submit a value in EUR, and the transform propagates the converted value in USD to the source fold. Multiple schemas—credit scoring, trade reporting, tax compliance—reference the same T_{fx}, ensuring consistent conversion logic across all of them.

Privacy-preserving feature engineering. A machine learning platform registers a suite of transforms: T_{clip} (value clipping), T_{bin} (histogram binning), and T_{noise} (calibrated noise addition). Each is irreversible and declares its label constraints. Researchers building training schemas compose these transforms in their fold definitions, selecting from the UTR catalog. Because each transform is content-addressed and pre-validated, a fold's privacy guarantees can be audited by inspecting the transform hashes alone—no need to review source code.

8.4 Relationship to the Fold Model

The UTR does not introduce a new trust boundary. Transforms remain bound to folds: a registered transform executes only when a fold invokes it, and only under the fold's access context. The UTR provides three additional guarantees:

- **Identity**: content-addressed hashes ensure that a transform referenced in a schema is exactly the transform that was validated.
- **Consistency**: all folds referencing the same hash produce identical transform behavior.
- **Auditability**: the UTR log records who registered each transform, when, and with what properties, enabling compliance review without inspecting individual fold definitions.

9 Related Work

Fold DB draws on several established lines of research:

- **Information flow control.** Denning's lattice model [3] provides the foundation for security labels. Fold DB applies lattice constraints at the transform level rather than at the language level.
- **Monadic computation.** Moggi's framework [6] supplies the semantic foundation. The reader-maybe composition captures policy-dependent computation with clean failure propagation.
- **Append-only logs.** Immutable data structures for audit and tamper-evidence appear in distributed ledgers [7] and log-based messaging systems [5]. Fold DB uses append-only storage for both data integrity and key-usage tracking.
- **Authenticated data.** Public-key signatures for write authentication follow standard constructions [8].
- **Cryptographic payments.** Integrating economic gates into data access extends Chaum's work on blind signatures [2] to per-query payment verification.
- **Database views.** Composable folds generalize the view mechanism in relational databases [1], adding field-level policy enforcement, transform composition, and economic controls.
- **Access control models.** Role-based access control (RBAC) assigns permissions to roles; attribute-based access control (ABAC) evaluates policies against request attributes. Fold DB's trust distance is closer to ABAC but uses a single scalar metric rather than arbitrary attribute predicates, trading expressiveness for simplicity and composability. Capability-based security, originating with Dennis and Van Horn (1966), grants unforgeable tokens for specific operations—Fold DB's cryptographic capabilities (WX_k, RX_k) adopt this pattern with the addition of bounded quotas.

10 Future Work

- **Multi-node federation**: extending trust distances and audit across a network of independent Fold DB nodes, with cross-node query routing and policy reconciliation.
- **Homomorphic transforms**: enabling third-party computation on encrypted fold outputs without revealing underlying data.
- **Zero-knowledge transform registration**: allowing new transforms to prove constraint compliance without revealing their internals [4].
- **Fold economy**: micropayment infrastructure for fold development and usage, creating economic incentives for high-quality transform libraries.

11 Conclusion

Fold DB enforces a simple invariant: data is accessible only through folds, and folds return only what the caller is authorized to see. Trust distance, cryptographic capabilities, security labels, and payment gates provide four independent, conjunctive layers of access control. The monadic semantics guarantee that policy failure at any point in a composed fold chain produces no output. We proved that this model satisfies data minimality—queries never exceed the authorized projection.

The current system operates on a single node. Extending it to federated multi-node computation, homomorphic transforms, and zero-knowledge transform registration are the primary directions for future work.

References

[1] Philip A. Bernstein and Nathan Goodman. Views and synchronization in databases. In *Proceedings of the 1977 ACM SIGMOD International Conference on Management of Data*, pages 1–8, 1977.

[2] David Chaum. Blind signatures for untraceable payments. *Advances in cryptology*, 82:199–203, 1983.

[3] Dorothy E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, 1976.

[4] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[5] Jay Kreps, Neha Narkhede, and Jun Rao. Kafka: a distributed messaging system for log processing. In *Proceedings of the NetDB*, volume 11, pages 1–7, 2011.

[6] Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.

[7] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008. `https://bitcoin.org/bitcoin.pdf`.

[8] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.