Datafold: A Unified Framework for Private Computation

Tom Tang

May 26, 2025

Abstract

Datafold secures computations on sensitive data without exposing raw inputs. It enforces explicit access policies, verifies user credentials and payments, and logs every operation. Users issue queries through "fold" interfaces, which apply controlled transformations and reveal only authorized results. Datafold's model meets privacy, traceability, and compliance requirements for healthcare, finance, and collaborative analytics.

1 Introduction

1.1 Motivation

Cloud and multi-tenant systems require secure, auditable data sharing. Traditional perimeter controls:

- Cannot prevent leaks during on-host processing.
- Lack fine-grained, transformation-level enforcement.
- Offer no cryptographic proof of privacy guarantees.
- Do not integrate economic controls into access.

Datafold isolates computations in verifiable "fold" contexts to address these gaps.

1.2 Notation

Define symbols used throughout:

- F: a fold (secure computation interface).
- D: raw input data.
- D': transformed output data.
- Q: query (derived from folds).
- $T : D \to \mathrm{F}[D']$: secure transformation.
- R: final computation result.
- π: minimal data revealed to the user.

- ℓ: security label for data fields.
- $W_n R_m$: write/read policy at trust distances n and m.
- $\mathcal{P}(u, F)$: payment verification predicate for user u on fold F.

Cryptographic Terms

- sk: private key.
- pk: public key.
- σ: signature.
- $\mathcal{D}$: decryption or safe reveal.

1.3 Related Work

Datafold builds on:

- **Append-Only Logs**: Immutable history for audit and rollback [9, 7].
- **Authenticated Writes**: Private-key signatures ensure data integrity [11].
- **Cryptographic Payments**: Tokens or signatures prove transaction completion [2].
- **Label-Based Flow Control**: Lattice-based policies prevent unauthorized flows [3].
- **Composable Views**: Virtual interfaces enforce data visibility rules [1].
- **Monadic Abstraction**: Structured computations with context-dependent side effects [8].

2 Core Architecture

Datafold uses *Fold* interfaces to encapsulate computations. Folds determine access control and payment logic. By adding new folds, access control can evolve over time to suit new use cases while the underlying data remains secure.

$$T(D) = D', \quad F(D') = \pi, \quad Q \text{ is derived from } F, \quad Q(\mathcal{C}) = \begin{cases} \text{Just}(\pi) & \text{if access policies and payment succeed} \\ \text{Nothing} & \text{otherwise} \end{cases}$$

The transformation T is applied to the underlying data D as it changes, producing an up-to-date derived value D'. The query Q is derived from the fold F, which evaluates which projection π of D' is authorized under the access context $\mathcal{C} = (u, \tau, \mathcal{L})$. If all policies and payments are satisfied, the projection π is returned; otherwise, the result is `Nothing`, ensuring no leakage.

2.1 Folds as Monads

A Fold encapsulates:

- Protected data values $V_i^{\ell_i, W_{n_i} R_{m_i}}$.
- A sequence of transformations $T_1 \gg\!= \cdots \gg\!= T_k$.
- Access rules based on AccessContext $= (u, \tau, \mathcal{L})$.

$$\text{FoldMonad}[a] = \text{AccessContext} \to \text{Maybe}\, a$$

2.2 Field-Level Policies

All field metadata (labels, trust distances) resides in the fold. Bind operations enforce policies at each step:

- Role-based views
- Field-level audit logs
- Secure composition of multiple folds

2.3 Composable Folds

Multiple folds can reference the same data value:

- Secure field reuse across domains.
- Derived computations by composing folds.
- Evolving access policies for the same data through different folds.

Folds remain direct, enforceable interfaces; explicit view layers are unnecessary.

2.4 Field Transform Selection

Each fold F produces a set of fields $\{f_1, f_2, \ldots, f_n\}$. Each field f_i is associated with a set of transformations $\mathcal{T}_i = \{T_{i1}, T_{i2}, \ldots\}$, and each transformation T_{ij} has a source fold $S_{ij} \in \mathcal{F}$.

For a given field f_i, only one transformation is selected for evaluation—specifically, the one whose source fold S_{ij} is registered for that field:

$$\text{ActiveTransform}(f_i) = \begin{cases} T_{ij} & \text{if } S_{ij} \text{ is registered for } f_i \\ \text{None} & \text{otherwise} \end{cases}$$

Subject to constraint:

$$\forall f_i,\ \exists{\leq}\, 1\ T_{ij} \in \mathcal{T}_i \text{ such that } S_{ij} \text{ is registered}$$

At query time, if an access context $\mathcal{C}$ is valid and the transformation T_{ij} is selected, the field resolves to:

$$f_i(\mathcal{C}) = T_{ij}(S_{ij}(\mathcal{C}))$$

This model supports deterministic, per-field derivation from composable folds while ensuring single-source lineage and policy isolation.

2.5 Recursive Fold Dependencies

Transforms depend on other folds, creating a recursive structure of evaluations. This enables higher-order analytics where folds compose over other policy-bound computations. Independent folds fields do not have transformations.

Let F_1, F_2, and F_3 be folds such that:

$$F_3 = \lambda \text{ctx}.\, F_2(\text{ctx}) \gg\!= T_3, \quad F_2 = \lambda \text{ctx}.\, F_1(\text{ctx}) \gg\!= T_2$$

Recursive folds allow:

- Modular reuse of validated sub-computations.
- Policy-preserving aggregation of multi-source data.
- Fault isolation—failure of one fold does not leak intermediate states.

2.6 Transformation Policies

Reversible and Irreversible transforms differ in policy.

- Reversible: writable, readable, applies in reverse when written, synchronizing upstream folds.
- Irreversible: readable, is not writable.

2.7 Access Control

Access is governed by trust distance τ, which reflects explicit trust proximity to the data owner. Each field carries explicit read/write bounds:

$$W_n R_m : \quad \text{Writable up to trust level } n, \quad \text{Readable up to trust level } m$$

Additionally, fields may carry cryptographic capability constraints:

- $\text{WX}_n(pk)$: Exclusive write permission granted to the holder of public key pk. Allows n writes; the counter decrements with each use.
- $\text{RX}_n(pk)$: Readable by any holder of pk. The read counter decrements after each access and enforces bounded disclosure.

2.8 Payment Model

Economic controls integrate payment verification:

$$C_{\text{linear}}(\tau) = a + b\,\tau, \quad C_{\text{exp}}(\tau) = a\,e^{b\,\tau}.$$

Lower trust distances incur lower costs.

3 Execution Model

3.1 System Model

Queries bind a fold to transformations under an access context:

$$Q = F \gg\!= T, \quad Q(\text{AccessContext}) = \begin{cases} \text{Just}(\pi) & \text{if policies hold} \\ \text{Nothing} & \text{otherwise} \end{cases}$$

3.2 Node Architecture

A Datafold node provides:

- **Registry**: Fold definitions, metadata, policies.
- **Execution Engine**: Runs fold computations under context.
- **Transform Library**: Vetted functions with complexity checks.
- **Append-Only Store**: Immutable data logs.
- **Audit Service**: Records all access, payments, and transformations.

4 Security Model

4.1 Threat Model

Datafold protects against:

- Unauthorized access
- Data tampering
- Information leakage
- Repudiation violations

4.2 Failure Modes

On policy or payment failure, evaluation returns `Nothing` and logs the event. No partial results leak.

4.3 Encryption at Rest

All data at rest—including fold definitions, metadata, and audit logs—is encrypted using authenticated encryption (AEAD) with keys derived from a master secret via HKDF or scrypt. Each dataset has a unique key. The append-only store logs all key usage to support revocation and enforce policy-compliant encryption scopes and rotation schedules.

5 Future Directions

Planned improvements extend the framework's utility and adoption:

- **Network Layer**: Enable secure, scalable, and privacy-preserving multi-node computation.
- **Homomorphic Transformations**: Allow secure third-party computations without revealing data.
- **Zero-Knowledge Transform Registration**: Add privacy-preserving proofs for new transformations to validate function constraints without revealing internals.
- **Economic Model for Fold Development**: Support micropayments for fold usage and development.

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] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography Conference (TCC)*, pages 265–284, 2006.

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

[6] T. Imieliński and W. Lipski. The relational model of data and cylindric algebras. *Journal of Computer and System Sciences*, 28(1):80–102, 1984.

[7] 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.

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

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

[10] John C. Reynolds. Compositionality in programming languages. In *Proceedings of the 8th International Symposium on Theoretical Aspects of Computer Software (TACS)*, pages 1–24, 1996.

[11] 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.