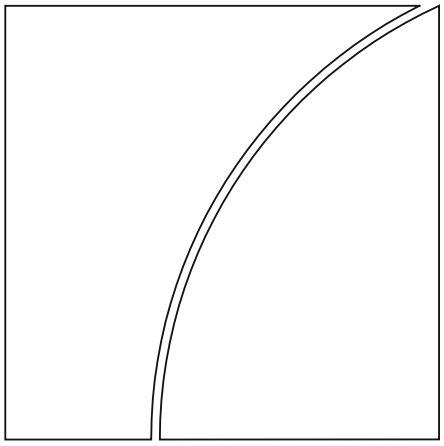


BIS Working Papers

No 1374

Verifiable official statistics: a blockchain-based approach

by Mario Rusev, Rafael Schmidt, Edward Lambe, Christian Schmieder and Glenn Philip Tice

Monetary and Economic Department

September 2026

JEL classification: C82, C88, G28, O33

Keywords: blockchain, SDMx, data integrity, cryptographic hashing, data dissemination

BIS Working Papers are written by members of the Monetary and Economic Department of the Bank for International Settlements, and from time to time by other economists, and are published by the Bank. The papers are on subjects of topical interest and are technical in character. The views expressed in this publication are those of the authors and do not necessarily reflect the views of the BIS or its member central banks.

This publication is available on the BIS website (www.bis.org).

© *Bank for International Settlements 2026. All rights reserved. Brief excerpts may be reproduced or translated provided the source is stated.*

ISSN 1020-0959 (print)
ISSN 1682-7678 (online)

Verifiable official statistics: a blockchain-based approach

Mario Rusev, Rafael Schmidt, Edward Lambe, Christian Schmieder and Glenn Philip Tice[1]

Abstract

International organisations including the Bank for International Settlements (BIS) have adopted SDMx (Statistical Data and Metadata) as the standard for exchanging official statistics. Trust in published data is essential for evidence-based policymaking. This paper shows how binding each SDMx dataset to its source using blockchain technology can enhance confidence in official statistics. We present a proof of concept implemented on the XRP Ledger (XRPL) and contribute, as an integrated whole, (i) an SDMx-native canonicalisation and per-`<Series>` hashing pipeline; (ii) a domain-separated Merkle aggregation scheme for batched anchoring; (iii) a self-contained, identity-bound verification artefact in which the SDMx message itself carries both the ordered Merkle leaves and a W3C Verifiable Credential signed by a publisher identity key cryptographically bound to the publisher's XRPL address via an on-chain attestation registry, so that any consumer can re-derive the anchored root and verify the publisher's identity from the file alone plus a single ledger lookup; (iv) an open-source XRPL-based reference implementation; and (v) a cost model that captures the batch size / latency / fee trade-off and is solved for an economically optimal batch size. The system enables near real-time data verification, provides cryptographic integrity guarantees, and establishes a foundation for future extensions, including zero-knowledge proofs and automated verification by AI agents. Measurements on the prototype show median publication latency of 3–5 seconds and verification latency of 1–2 seconds under the controlled test conditions. The approach is data-format-agnostic and can be extended to other structured statistical or regulatory formats.

Keywords: Blockchain, SDMx, data integrity, cryptographic hashing, data dissemination

JEL classification: C82, C88, G28, O33

[1] Mario Rusev works for d-fine Austria GmbH. The other authors work for Bank for International Settlements. We would like to thank Bruno Tissot, Olivier Sirello, Johannes Damp, Stephan Probst and Simon Papenkort for their helpful advice and collaboration. The authors acknowledge the use of generative AI for language editing purposes. The authors bear full responsibility for the factual accuracy of this paper. References to individual firms and projects are for illustrative purposes and should not be construed as any statement on the soundness or the legal or regulatory status of any firm or project.

Contents

1. Introduction

The authenticity, integrity, reproducibility and trustworthiness of official statistics are foundational for evidence-based policymaking, financial stability analysis and public accountability (United Nations, 2023a). The economic value of trust in official statistics is substantial and increasingly recognised. While an exact quantification is difficult, an event-study analysis published by the NBER (Bloom et al, 2026) relating to market uncertainty following the August 2025 dismissal of the head of the US Bureau of Labor Statistics has been cited to suggest that erosion of trust in official statistics can carry significant economic costs. This motivates mechanisms that make data integrity independently verifiable, even if precise quantification of the benefits remains an active research question.

SDMx (Statistical Data and Metadata) is an international standard and technical infrastructure sponsored by the world's leading international economic, financial and statistical organisations – including the Bank for International Settlements (BIS), the European Central Bank (ECB), the Statistical Office of the European Union (Eurostat), the International Monetary Fund (IMF), the Organisation for Economic Co-operation and Development (OECD), the United Nations (UN), the World Bank Group and the International Labour Organization (ILO) – designed to streamline the exchange, processing, and dissemination of statistical data and metadata.

Statistical authorities, including central banks, increasingly rely on automated and standardised information systems to disseminate their statistical outputs. While platforms such as SDMx provide a robust framework for data exchange[2], they lack native cryptographic capabilities to verify data provenance and detect unauthorised modifications, which can compromise the integrity of statistical information.

This gap poses challenges for both users and producers of trusted statistical data. For users, the rapid proliferation of heterogeneous alternative sources makes it increasingly difficult to distinguish authoritative information from unreliable content. This challenge is further exacerbated by artificial intelligence (AI) systems which, in the absence of rigorous quality frameworks, often cannot ensure the provenance and reproducibility of statistical data (United Nations, 2023b). For producers, the priority is to develop innovative, cost-effective solutions that safeguard the authenticity and integrity of the data they disseminate (Ricciato et al, 2023).

Our approach addresses the key challenge – preserving authoritative information that is authentic and has demonstrable integrity – by combining blockchain with well established statistical standards such as SDMx. Unlike solutions based solely on Content Credentials or traditional digital signatures, it provides a decentralised, tamper-evident audit trail that does not rely on a single authority and supports real-time verification. The system meets key business needs of statistical authorities and, more broadly, of other authoritative data producers, including regulatory compliance, auditability and trusted data sharing across organisations.

[2] SDMX: Statistical Data and Metadata eXchange, official site, https://sdmx.org/.

The primary contribution of this paper is a production-oriented, SDMx-native mechanism for cryptographically binding published official statistics to their source, enabling independent verification without altering existing dissemination workflows.

To make the novelty explicit relative to prior data-verification and blockchain work, our contributions are as follows.

1. **SDMx-native fingerprinting.** A canonicalisation and hashing pipeline that operates at both whole-file and per-<Series> granularity, so publishers can anchor a single dataset, a subset of series, or an entire release with the same machinery (Section 4).
2. **Domain-separated Merkle aggregation.** An explicit leaf/node construction with byte-level domain separators and length prefixes that prevents the concatenation-ambiguity and second-preimage attacks that affect naive Merkle implementations (Section 4.1).
3. **Self-contained, identity-bound verification artefact.** The publication step rewrites the SDMx `<message:Source>` header to carry (a) a structured anchor descriptor (`Transaction Hash:...; Type:Partial; Root:...; Leaves:N`), (b) an ordered `MerkleLeaves:...` list of per-series fingerprints, and (c) a base64-encoded W3C Verifiable Credential signed by a publisher identity key (Ed25519) cryptographically bound to the XRPL address via an on-chain attestation registry, binding the on-chain transaction hash, root and per-series fingerprints to the publisher's identity (`did:xrpl:<address>`). Verification re-derives the root from the file's own `MerkleLeaves`, re-canonicalises and matches it against the on-chain memo, and cross-checks the embedded credential's `publicKeyHex` against the transaction sender's address `tx_meta.Account` (via the on-chain attestation registry). A consumer therefore learns who published the data and that the bytes are unchanged from the file alone plus a single ledger lookup, with no dependency on the publisher's backend.
4. **Open-source XRPL reference implementation.** A containerised proof of concept with REST APIs that statistical organisations can deploy alongside existing SDMx endpoints. The code is publicly available and is being published as a prototype through BIS Open Tech, a platform for sharing statistical and financial software as public goods[3], and through the SDMx community site.[4]
5. **Cost model with an economically optimal batch size.** A closed-form model that captures the on-chain fee, storage, processing and waiting-cost components and admits an explicit optimum b^* for batched anchoring, including a multi-class extension for mixed-urgency workloads (Section 6.3).

Individually, hash anchoring, Merkle trees and verifiable credentials are well known; the contribution lies in composing them around the SDMx information model and quantifying the resulting operating regime. Table 1 positions this contribution against the closest prior efforts (Statistics Canada (Klein et al, 2022), DOC/Chainlink (Chainlink, 2025), and traditional PKI / Content Credentials), summarising for each whether they cover (i) SDMx semantics, (ii) batched Merkle anchoring, (iii) verifiable-

[3] "BIS Open Tech", a platform for sharing statistical and financial software as public goods, https://www.bis.org/innovation/bis_open_tech.htm.

[4] SDMX: "SDMX tools and community resources", https://sdmx.io/.

credential binding, (iv) an explicit cost / batching model, and (v) an open reference implementation.

Positioning of this work against representative prior efforts Table 1

	SDMx semantics	Merkle batching	VC binding	Cost model
StatCan (Klein et al, 2022)	partial	no	No	no
DOC/Chainlink (Chainlink, 2025)	no	no	No	no
PKI / Content Credentials	partial	no	Partial	no
This work	**full**	**full**	**Full**	**full**

Coverage of (i) SDMX semantics, (ii) batched Merkle anchoring, (iii) verifiable-credential binding, (iv) an explicit cost / batching model and (v) an open reference implementation.

Sources: Authors

The canonical idea of timestamping data for later verification dates back to the early work on cryptographic time-stamping (Haber and Stornetta, 1991); modern distributed ledgers and Merkle tree constructions make these concepts practical at scale (Buldas and Saarepera, 2004; Zheng et al, 2017; Nakamoto, 2008]. Several initiatives have explored blockchain-based solutions for data authentication, including Statistics Canada's investigation into publishing dataset hashes on distributed ledgers (Ricciato et al, 2023). Recent research has further demonstrated the practical benefits of blockchain technology for enhancing trust in official statistics (Ricciato et al, 2023). However, prior implementations have focused primarily on simple hash anchoring without comprehensive SDMx integration or real-time verification capabilities. The United States Department of Commerce (DOC) has worked together with Chainlink to bring macroeconomic data from the US government's Bureau of Economic Analysis (BEA) on-chain (Chainlink, 2025). These Chainlink Data Feeds securely deliver critical information around key US economic indicators on-chain, including real gross domestic product (GDP), the personal consumption expenditures (PCE) price index and real final sales to private domestic purchasers.

Our work builds on these foundations by creating a standards-compliant, SDMx-aware system for anchoring and verifying datasets at scale, and by explicitly addressing practicalities such as canonicalisation, domain-separated hashing for Merkle trees, inclusion proofs and integration with verifiable credentials. Specifically, a particular type of blockchain – the XRP Ledger (XRPL) – has been used as a proof of concept because of its low nominal fees, fast consensus finality, availability of developer resources and technical analysis of the consensus protocol (Chase and E MacBrough, 2018).[5] Yet our approach is meant to be blockchain-agnostic and to work with solutions other than XRPL as well.

We present a complete and operational system that:

- integrates with existing SDMx workflows and tools with minimal disruption;

[5] See also XRPL.org: "XRP Ledger documentation and developer resources", https://xrpl.org/docs.

- provides real-time verification capabilities for end users and automated agents, maintaining compatibility with SDMx and metadata requirements;
- leverages XRPL for cost-efficient, rapid transaction anchoring;
- demonstrates prototype-level performance with median publication latencies of 3–5 seconds in the test environment described in Section 7 – we are explicit that this characterises the proof of concept, not a fully hardened production deployment; and
- offers a practical pathway to advanced features such as zero-knowledge proofs and selective disclosure.

The business value proposition is substantial: organisations can provide cryptographic proof of data integrity while leveraging the interoperability capabilities offered by SDMx. This enables new use cases such as integration for real-time data oracles and automated compliance verification for regulatory reporting, the issuance of inflation-linked products (or smart contracts), transparent dashboards powered by immutable data, the integration of perpetual futures and the issuance of new types of digital asset. For example, verified SDMx data can enable the creation of programmable digital assets such as tokenised bonds and derivatives that automatically adjust their terms based on economic indicators (eg GDP growth or inflation rates), offering advantages over traditional inflation-linked instruments in terms of composability, transparency and automated execution (QuickNode, 2025). Similarly, transparent dashboards for public policy monitoring (such as the IMF Climate Change Indicators Dashboard[6]) can leverage verifiable data to allow users to audit the underlying statistics independently, enhancing trust in evidence-based policymaking.

The remainder of this paper is organised as follows. Section 2 provides an intuitive primer on the cryptographic building blocks for readers from statistics or policy backgrounds. Section 3 describes the system architecture and components. Section 4 details the cryptographic workflow and formal properties, and Section 5 walks through a concrete worked example end to end. Section 6 presents the cost model and a closed-form economically optimal batch size, including a multi-class extension. Section 7 presents the performance evaluation, including the test environment and the methodology behind the comparison table. Section 8 discusses use cases and an econometric anomaly-detection example for dataset tampering. Section 9 provides the formal foundations – threat model, a Merkle binding theorem, an on-chain footprint lower bound, a BBS+ selective-disclosure extension, a multi-publisher accumulator, and a strategic-batching game with a VCG mechanism. Section 10 covers security, privacy, governance and long-term operational risks (key management, revocation, errors, blockchain availability and fork scenarios). Section 11 discusses applicability beyond SDMx. Section 12 introduces considerations for AI agents and automated verification workflows. Section 13 concludes.

[6] International Monetary Fund (2025): "IMF data dashboards", https://data.imf.org/en/Dashboards.

2. An intuitive primer on the building blocks

Before turning to the formal description, we briefly explain in plain terms the four cryptographic ideas on which the rest of the paper relies.

- **Hashing as a digital fingerprint.** A cryptographic hash function (we use SHA3-512) maps a file of arbitrary size to a short, fixed-length string of bits. Two important properties hold: the same input always produces the same fingerprint, and changing even a single byte of the input produces a completely different fingerprint. The fingerprint reveals nothing about the contents, but anyone who has the original file can recompute the fingerprint and check that it matches.
- **Merkle aggregation as a tree of fingerprints.** If a publisher has many datasets, it would be wasteful to record every fingerprint individually on a blockchain. A Merkle tree solves this: fingerprints are paired and re-hashed in successive layers until a single "root" fingerprint at the top represents the entire batch. To prove that a particular dataset belongs to the batch, the verifier only needs the dataset itself plus a short "inclusion proof" (a logarithmic-size list of sibling hashes). One root recorded on-chain therefore commits to thousands of datasets at once.
- **Blockchain anchoring as a public timestamped notary.** "Anchoring" means writing the Merkle root into a transaction on a public blockchain (here, the XRP Ledger). Once the transaction is included in a ledger version, the root is timestamped and effectively immutable: no party can alter the recorded fingerprint after the fact without rewriting the entire ledger, which is computationally and economically infeasible. The blockchain therefore plays the role of a public, decentralised notary, with no single institution able to edit the record silently.
- **Verifiable credentials as digitally signed badges.** A W3C Verifiable Credential is a tamper-evident JSON object signed by an issuer (e.g., a statistical authority) that asserts statements about a subject (e.g., "the dataset with fingerprint h was published by us on date t"). Bundling the dataset with its credential and Merkle inclusion proof lets a consumer check, offline, both the *identity* of the publisher and the *integrity* of the bytes.

A reader who keeps these four pictures in mind – fingerprint, tree, notary, signed badge – can follow the rest of the paper without further cryptographic background.

Section summary. The system rests on four ideas: hashing produces a tamper-evident fingerprint of a dataset; Merkle aggregation collapses many fingerprints into one root; blockchain anchoring publishes that root in a public, timestamped, append-only ledger; and verifiable credentials bind the fingerprint to a known publisher. The remainder of the paper formalises how each of these is applied to SDMx.

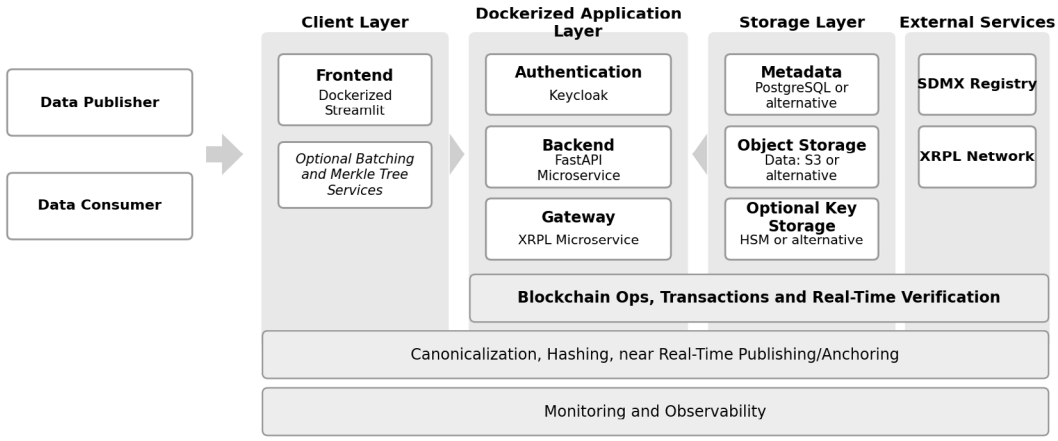
3. System architecture and components

Graph 1 illustrates the high-level architecture of our blockchain-enhanced SDMx proof-of-concept dissemination platform. The implementation follows a microservices approach with components deployed as Docker containers (Merkel,

2014), ensuring portability and reproducible deployments. In a production scenario, the data consumer would verify the integrity of the published data using its own software stack and connection to the immutable shared ledger. The architecture is organised into four main layers – client, storage, external services and blockchain operations – that together provide a comprehensive solution for verifiable data dissemination. Each layer is decoupled behind a narrow interface, so individual components (identity provider, object store, target blockchain) can be swapped without touching the rest of the system. The remainder of this section describes each layer and the responsibilities of its components, mapped to the prototype's Docker deployment.

Simplified system architecture of the blockchain-enhanced SDMx dissemination platform

Graph 1



Source: Authors

Client layer

This layer encompasses both data publishers and consumers, providing user interfaces and application logic. It includes:

- **Backend service (FastAPI).** Serves as the central orchestration layer, providing RESTful API endpoints for SDMx publication and validation. In production, this component would integrate with enterprise authentication systems and handle load balancing. The backend manages the entire workflow from XML canonicalisation to blockchain transaction coordination.
- **Frontend interface (Streamlit).** Provides user-friendly access for both data publishers and validators. This component demonstrates the user experience and highlights the modularity of the solution, which can be tailored to enterprise-grade web applications in production deployments.
- **Identity and access management (Keycloak / DID).** Manages authentication and authorisation using OAuth2/OpenID Connect (Recordon and Reed, 2006). In production, this would integrate with organisational identity providers and

support advanced features such as decentralised identifiers (DIDs) for verifiable credentials.

- **XRPL integration gateway.** A specialised service handling all blockchain interactions, including transaction submission, memo anchoring and verification. This abstraction layer allows future migration to alternative blockchains or smart contract platforms.

Storage layer

This layer provides persistent storage for different types of data, ensuring data availability and integrity. It includes:

- **Data storage.** Stores anchor metadata, Merkle proofs, timestamps, transaction references and audit logs. Production deployments would implement high-availability configurations and comprehensive backup strategies.
- **Object storage (S3-compatible).** Handles raw SDMx dataset files, with potential integration with S3-compatible storage or alternative solutions such as IPFS for decentralised storage in production.
- **Key management service (optional HSM).** A separate security-critical component for managing cryptographic signing keys, which would integrate with hardware security modules (HSMs) in production environments.

External services

This layer integrates with external systems essential to the workflow, including:

- **SDMx registry.** Provides standardised metadata and data structure definitions, ensuring compliance with SDMx standards.
- **XRPL network.** Serves as the immutable ledger for anchoring dataset fingerprints, providing timestamped, tamper-evident commitments.

Blockchain operations

This layer handles the core cryptographic processes that ensure data integrity, including:

- **SDMx processing engine.** The core component responsible for canonical XML processing and cryptographic hashing. This ensures deterministic fingerprinting of SDMx data regardless of formatting differences.
- **Canonicalisation, hashing and Merkle tree construction.** Processes for standardising SDMx data, generating cryptographic fingerprints and aggregating multiple datasets through Merkle tree structures for efficient anchoring.
- **Near real-time publishing/anchoring.** Mechanisms for the timely submission of data commitments to the blockchain ledger.
- **Monitoring and observability.** Systems for tracking the health, performance and operational status of the entire publication and verification workflow.

Section summary. The architecture is a four-layer microservice deployment (client, storage, external services, blockchain operations). Each layer is replaceable:

the storage backend, identity provider and target blockchain are all behind narrow interfaces, which is what makes the system blockchain-agnostic and incrementally adoptable on top of existing SDMx infrastructure.

4. Cryptographic workflow: publication and verification

This section formalises the canonicalisation, hashing, Merkle aggregation and on-chain anchoring processes, and states precise domain-separation rules to avoid ambiguous encodings. Readers who are new to these primitives may first wish to consult the intuitive primer in Section 2.

4.1 Canonicalisation and hashing

For deterministic hashing we apply XML canonicalisation (Canonical XML 1.1) to the SDMX-ML payload X as specified by the W3C recommendation (Boyer and Davis, 2008). Because the publication step (Algorithm 1) later injects anchor descriptors and verifiable credentials as <message:Source> elements carrying sourceID="sji:anchor", and because a file may accumulate several such anchors over successive anchoring runs, the fingerprint must commit to the underlying statistical payload alone – independently of any anchor history. We therefore define an idempotent anchor-stripping projection

$$strip_{anchor}(X) = X \text{ with every } < message{:}Source > \text{ element s.t. sourceId = "sji:anchor" removed,}$$

and canonicalise the stripped document:

$$\tilde{X} = C14N(strip_{anchor}(X))$$

The canonicalised byte string $\tilde{X}$ is hashed using SHA3-512 (FIPS 202) (National Institute of Standards and Technology, 2015):

$$h = SHA3-512(\tilde{X}) \in \{0,1\}^{512}$$

Since $strip_{anchor}$ removes all anchor elements regardless of their `tx_hash`, and is idempotent ($strip_{anchor} \circ strip_{anchor} = strip_{anchor}$), the fingerprint h is invariant under the addition or removal of any number of anchors. Verification is thus deterministic and independent of a file's anchor history, whether it carries zero, one or many anchors from prior or future anchoring runs. (Per-`<Series>` fingerprints below are computed over `<Series>` subtrees, which never contain anchor elements, so the same guarantee holds trivially for them.)

Remark (canonical forms and gauge invariance). An SDMX-ML payload X does not possess a unique byte representation; it is an equivalence class of semantically identical XML documents. Canonicalisation (C14N 1.1) acts as a gauge choice, projecting this equivalence class onto a single canonical representative $\tilde{X}$. Hashing $\tilde{X}$ yields a cryptographic commitment that is invariant under syntactic perturbations, ensuring that verification depends only on the semantic content of the data.

Deterministic processing. Using C14N ensures that differences in insignificant whitespace, attribute ordering or namespace prefixes do not change the fingerprint, providing consistent hashing across XML processors.

Series-level hashing. In addition to whole-file fingerprints, the system supports anchoring at the granularity of individual `<Series>` elements. Each `<Series>` element S_i is canonicalised (C14N) and hashed using SHA3-512 to produce a per-series fingerprint

$$\boldsymbol{h_i = SHA3 - 512(C14(S_i))}$$

These per-series fingerprints h_i are treated as the logical dataset identifiers (leaves) for Merkle aggregation when publishers choose to anchor a selection of series rather than the entire file. This approach is illustrative and can easily be extended to other elements, such as observations of series.

Domain separation and pair encoding. To prevent ambiguous combinatorial collisions in the Merkle tree, we use explicit domain separators and length-prefixing when composing nodes. We recommend the following concrete construction:

$$\boldsymbol{H_{leaf}(h) = SHA3 - 512(0x00||h)}$$

$$\boldsymbol{H_{node}(L,R) = SHA3 - 512(0x01||L||R)}$$

where `0x00` and `0x01` are single-byte domain separators. Because all hashes are fixed-length 64-byte SHA3-512 outputs, the separators alone suffice to prevent concatenation ambiguity between leaves and nodes; the $len32(\cdot)$prefix is not needed for the fixed-length leaf and internal node concatenations, but is explicitly required in the final root wrapper to bind the leaf count n. For n > 1, the leaf vector is padded to the next power of two ($2^{\lceil \log_2 n \rceil}$) by duplicating the final leaf, forming a perfect binary tree. For n = 1, no padding is needed (the tree is a single leaf). To prevent leaf-duplication ambiguity additionally (e.g., n = 3 padded to four leaves by duplicating h_3, and n = 4 with $h_4 = h_3$, would produce the same internal root), the final root wraps the internal root with a domain-separated layer that explicitly binds the leaf count via len32(n):

$$\boldsymbol{R = SHA3 - 512(0x02||len32(n)||Root_{internal})}$$

where len32(n) is the 32-bit big-endian encoding of the number of leaves n.

4.2 Merkle aggregation and proofs

Given n leaf fingerprints $h_1, \ldots, h_n$ (each a 512-bit string), where each hi may be a per-series fingerprint produced as above, leaves for the Merkle tree are computed as

$$L_i = H_{leaf}(h_i)$$

Internal nodes follow the domain-separated scheme above. The Merkle root R is the topmost node value. A consumer possessing the full leaf vector recomputes R directly and compares it with the on-chain root; no inclusion proofs are required for standard verification.

Inclusion proofs are generated on demand for selective-disclosure scenarios – for example, when a consumer wishes to prove a subset of series to a third party without transmitting the entire file. An inclusion proof for leaf i- is the ordered list of

sibling node values $\pi_i = (s_1, \ldots, s_m)$ together with orientation (left/right); verification recomputes the root by iteratively applying $H_{node}(\cdot,\cdot)$.

Proof verification (selective disclosure). Let $root(h_i, \pi_i)$ denote the recomputed root. The verifier checks whether $root(h_i, \pi_i) = R$. This recomputation requires the leaf count n- (implicitly provided by the `Leaves:N` field in the SDMx anchor descriptor) to compute the final domain-separated wrapper $\boldsymbol{R = H(0x02||len32(n)||Root_{internal})}$. We emphasise that proof objects include orientation bits (left/right) or an ordered pair convention so that the recomputation is unambiguous.

4.3 Blockchain implementation alternatives

While our current implementation uses XRPL memo fields for lightweight anchoring, production systems could leverage smart contracts on EVM-compatible sidechains for enhanced functionality. Smart contracts would enable more complex features such as:

- access control and permissioning mechanisms;
- automated compliance checking through on-chain logic;
- multi-signature requirements for sensitive data;
- integration with decentralised oracle networks such as Chainlink; and
- programmable data validation rules.

For specific use cases such as the issuance of inflation-linked products, perpetual futures or other tokenised financial instruments, having both data anchoring and smart contract execution on the same blockchain offers significant advantages. This unified approach enables seamless integration of verifiable SDMx data directly into smart contract logic, allowing for automated, regulatory-compliant financial instruments that can programmatically verify the integrity of their underlying data sources.

Our architecture supports several implementation strategies to address these needs:

- **Single-chain smart contract integration.** Using XRPL's EVM-compatible sidechain (or some future native smart contract capabilities) allows both data anchoring and financial instrument execution on the same ledger, reducing cross-chain complexity while maintaining regulatory compliance through on-chain verification of data integrity.
- **Multi-chain publication.** For enhanced redundancy and interoperability, the same data anchors can be published across multiple blockchains (eg XRPL for cost efficiency, Ethereum for DeFi integration, and private/permissioned chains for regulated applications). This multi-chain approach ensures data availability across different ecosystems while maintaining consistent verification proofs.
- **Decentralised oracle services.** Integration with oracle networks such as Chainlink would allow SDMx data to be reliably bridged to any blockchain, enabling smart contracts on various platforms to access verified statistical data.

This creates a flexible ecosystem in which the anchoring blockchain can be optimised for cost and performance, while consumption occurs where needed.

- **Hybrid approach.** A practical solution might combine XRPL's efficient anchoring with smart contract execution on compatible chains, using cross-chain messaging protocols (such as IBC, Wormhole or Chainlink CCIP) (Herlihy, 2018) to maintain consistency and enable regulatory compliance across the entire system.

The current memo-based approach favours operational simplicity and minimal costs, while smart contract integration offers greater functionality for financial applications at the expense of increased complexity and gas costs. For enterprise deployments, the choice between these approaches would depend on specific regulatory requirements, integration needs and the complexity of the use cases being addressed.

For financial use cases requiring strict regulatory compliance, smart contracts provide verifiable, transparent execution environments that can embed compliance rules directly into contract logic. This ensures that only verified, anchored data are used for critical functions such as calculating inflation adjustments, triggering derivative settlements or determining interest payments.

In summary, our system's blockchain-agnostic design supports flexible deployment strategies ranging from simple single-chain anchoring to sophisticated multi-chain architectures with smart contract integration. This adaptability ensures that organisations can choose the optimal implementation for their specific needs, whether prioritising cost efficiency, regulatory compliance or integration with existing financial ecosystems.

Section summary. Each SDMx payload is canonicalised (C14N 1.1), hashed with SHA3-512 at whole-file and per-`<Series>` granularity, aggregated through a domain-separated Merkle tree, and anchored on-chain as a single root. Verification reduces to recomputing the root from the dataset and a logarithmic-size inclusion proof and comparing it with the on-chain value. The blockchain layer is intentionally interchangeable.

5. Worked example: publishing and verifying one SDMx dataset

To make the workflow concrete for an operations team, we walk through a single, realistic dataset end to end. We use a small SDMX-ML 2.1 `StructureSpecificData` message containing one dataflow (`BIS.CBS`, consolidated banking statistics) with three `<Series>` elements (three reporting countries) and 12 quarterly observations each. The file is approximately 24 KB on disk.

Step 1 – Submission. The publisher's SDMx dissemination tool sends an HTTP POST to `/publishSDMX` on the backend service, with the SDMX-ML message as the request body and an OAuth2 bearer token (issued by the Keycloak realm) identifying the publisher and carrying the `wallet_admin` role required to draw on the institutional anchoring wallet.

Step 2 – Canonicalisation. The backend first strips every `<message:Source>` anchor element (`sourceID="sji:anchor"`) that a prior anchoring run may have left in the file, then applies Canonical XML 1.1 to the anchor-free message, producing $\tilde{X} = C14N(strip_{anchor}(X))$. Whitespace, attribute ordering and namespace prefixes are normalised so that semantically identical XML always yields the same bytes, and the resulting fingerprint commits to the statistical payload alone – independently of how many times the file has been anchored.

Step 3 – Hashing. The backend computes the whole-file fingerprint $h = SHA3-512(\widetilde{X})$ - and, in parallel, the three per-series fingerprints h_1, h_2, h_3 by canonicalising each `<Series>` subtree independently. In our run, h begins `0x4f3a...` (truncated for display).

Step 4 – Merkle aggregation. Because the publisher chose to anchor at the series level, the three fingerprints become leaves $L_i = H_{leaf}(h_i)$. The leaf vector is padded to the next power of two ($2^{\lceil \log_2 3 \rceil} = 4$) by duplicating the final leaf, forming a perfect binary tree of height 2 whose internal root is $R_{int} = H_{node}(H_{node}(L_1, L_2)H_{node}(L_3, L_3))$. The final domain-separated root is $\boldsymbol{R} = \boldsymbol{H(0x02||len32(n)||Root_{int})}$. The inclusion proof for series 1 is $\pi_1 = (L_2, H_{node}(L_3, L_3))$ with orientation bits (R, R)-.

Step 5 – Anchoring on XRPL. The XRPL integration gateway constructs an XRPL `Payment` transaction from the institutional anchoring wallet to a designated destination address, with a minimal economic value (10 drops, ie 10^{-5} XRP) sufficient only to cover ledger acceptance. Its `Memos` field carries the raw hexadecimal encoding of R as the sole memo payload, with no in-band prefix or framing – the structured anchor descriptor lives in the SDMx file rather than in the memo, keeping the on-chain footprint minimal and format-independent. The transaction is signed and submitted via JSON-RPC to a public XRPL DevNet node; after a typical ledger close (3–4 seconds) the transaction is included and `tx_hash` is returned. The backend then rewrites a single placeholder `<message:Source>` element inside the returned SDMx message to `Transaction Hash:<tx_hash>; Type:Partial; Root:<hex(R)>; Leaves:N; MerkleLeaves:<h1,h2,...,hN>`, carrying both the anchor metadata and the ordered leaf fingerprints in one element. The returned file is therefore self-contained: it carries within itself both the anchored root and every input required to recompute it.

Step 6 – Verifiable-credential issuance. The backend constructs a W3C Verifiable Credential whose `credentialSubject` binds the dataset identifier, the on-chain transaction hash, the root R, the anchor type (`Partial` or `WholeFile`) and the ordered per-series fingerprints. The credential is serialised as a JSON-LD document and canonicalised using the RDF Dataset Canonicalization algorithm (URDNA2015) per the W3C `Ed25519Signature2020` suite specification. It is signed with the publisher's dedicated identity signing key, cryptographically separate from the institutional XRPL anchoring key (Ed25519). The publisher registers this identity key against its XRPL address in an on-chain attestation registry – a lightweight XRPL hook that maps each anchoring address to the set of authorised credential signing public keys, updatable by a signed registration transaction from the anchoring key. The resulting proof block (`Ed25519Signature2020`) includes the publisher's `publicKeyHex` and a `did:xrpl:<address>` verification method. The signed credential is base64-encoded and embedded as an additional `<message:Source>` element (`VC:...`) inside

the returned SDMx message, so the publication response is itself the receipt (R, `tx_hash`, `ledger_index`, VC) rather than an out-of-band side channel.

Step 7 – Consumer verification. A consumer who has downloaded the returned SDMx message calls `/validateSDMX` (or runs the verification client locally). The verifier parses the `<message:Source>` header to extract `tx_hash`, the ordered `MerkleLeaves` and the embedded credential; then (a) re-canonicalises each `<Series>` subtree to recompute the per-series fingerprints and confirms that the declared leaves are exactly those present in the file; (b) reconstructs the Merkle root R′ from the declared leaves using the same domain-separated scheme; (c) fetches the referenced XRPL transaction via a public node and reads the memo; (d) verifies that the on-chain memo equals R′; and (e) verifies the credential signature and cross-checks the credential signing public key against the on-chain attestation registry for the XRPL address given by `tx_meta.Account` (the transaction sender; using `SigningPubKey` may differ under regular-key signing), by looking up the set of authorised credential signing keys registered by that XRPL address, as well as verifying the credential subject's `txHash` and `merkleRoot` against what was actually anchored. A pass on all five checks tells the consumer who published the data and that the bytes are unchanged, with no trust placed in the publisher's backend. End to end, observed publication and verification latencies fall within the 3–5 second and 1–2 second envelopes characterised in Section 7.

What changes for a thousand-dataset release? The only steps that scale with the batch are 3 (per-leaf hashing, linear in n) and 4 (tree construction, also linear). Steps 5–7 do not change: one transaction still anchors the whole batch, and per-dataset verification still requires only $\lceil \log_2 n \rceil$ sibling hashes.

Section summary. A single SDMx dataset traverses seven concrete steps from submission to consumer verification. Anchoring cost is independent of batch size; per-dataset verification cost is logarithmic in the batch.

6. Cost model

We model three main cost components for each dataset:

- on-chain anchoring cost (transaction fees): C_{chain}
- off-chain storage and retrieval cost (object store, eg S3): C_{store} and
- processing and orchestration cost (compute, network, monitoring): C_{proc}

Let f denote the (nominal) XRPL transaction fee (in XRP) and ρ the currency conversion to fiat. For a batch of size $b$anchored with one transaction, the on-chain fee per dataset is

$$C_{chain} = \frac{f}{b}\rho$$

Storage cost per dataset. Storage costs have three components with different dependencies on the batch size b.

- **Raw dataset storage:** $s \cdot \sigma(S_{total})$- (where $\sigma'(S_{total}) < 0$-, approximated as $\approx s \cdot \bar{\sigma}$,, where s is dataset size and $\bar{\sigma}$is the (constant) simplified storage price per MB.

This cost is independent of batch size because each dataset's raw content must be stored individually. The raw dataset storage cost for a dataset of size s is approximately$s \cdot \sigma(S_{total})$, where S_{total}- is the total storage volume and $\sigma(S)$- is a decreasing function reflecting volume discounts in cloud storage. In practice, cloud providers use tiered pricing where the marginal cost per MB decreases as total storage increases (economies of scale). For modelling purposes, we use the simplifying assumption $\sigma(S) \approx \bar{\sigma}$, a constant average storage cost, but note that in production deployments the effective per-dataset storage cost would decrease with the publisher's total data volume. The cost advantage of batching would therefore be slightly amplified, because larger publishers (who would naturally batch more) also get better storage pricing.

- **Merkle proof storage** (mandatory if partial verification is supported): $\frac{k}{2^{20}} \cdot \lceil \log_2 b \rceil \cdot \sigma_{proof}$, converting byte-size k to MB to match the units of σ_{proof}; the proof length $\lceil \log_2 b \rceil$ is 0 at b = 1 and non-negative for all feasible b ≥ 1, where:
    - k is the size per Merkle proof node (64 bytes for SHA3-512);
    - $\lceil \log_2 b \rceil$ is the Merkle proof length (number of hash values needed), 0 when b = 1 (a single-leaf tree has no proof); and
    - σ_{proof} is the storage cost per MB-month.
    - This per-dataset cost is mandatory when the API offers partial verification (any consumer may retrieve a single series' inclusion proof), which is the standard SDMX REST access pattern. It may be omitted only when the publisher guarantees that all consumers will always receive the complete leaf vector.
- **Fixed batch overhead:** $\frac{C_{batch\_fixed}}{b}$, where C_{batch_fixed} represents fixed one-time costs per batch (transaction metadata, indexing overhead) that are amortised across b datasets.

Storage costs recur monthly (or over the chosen retention horizon). Introduce a time horizon T (in months, e.g., the expected retention period) to convert per-month rates into total cost. The complete storage cost over T months is:

$$C_{store}(b, T) = T \cdot \left[s \cdot \bar{\sigma} + \frac{k}{2^{20}} \cdot \lceil \log_2 b \rceil \cdot \sigma_{proof} \right]$$

Processing cost per dataset. This includes CPU, canonicalisation, hashing, proof generation and database writes (a constant approximation): $C_{proc} = c$.

Total cost per dataset. The chain cost, processing cost and batch overhead are one-time expenses; the storage cost recurs over the retention horizon T months. The total cost per dataset over horizon T is:

$$\begin{aligned} C_{total}(b, T) &= C_{chain}(b) + C_{store}(b, T) + \frac{C_{batch\_fixed}}{b} + C_{proc} \\ &= \frac{f}{b} \cdot \rho + T \cdot \left[s \cdot \bar{\sigma} + \frac{k}{2^{20}} \cdot \lceil \log_2 b \rceil \cdot \sigma_{proof} \right] + \frac{C_{batch\_fixed}}{b} + c \end{aligned}$$

6.1 Reasoning for the storage cost structure

The storage cost model reflects our system's actual architecture.

- **Raw dataset storage** is batch-independent in our baseline design. In our current architecture, each SDMx dataset must be stored in its entirety for retrieval and verification. This creates a baseline storage cost that is independent of batch size because each dataset's raw content must be stored individually. This design choice reflects a conservative approach to data availability and auditability in enterprise environments. However, several alternatives exist that could modify this cost structure:
    - *Data compression*. Raw XML can be compressed with standard algorithms (gzip, Brotli, etc) achieving 70–90% size reduction, effectively reducing s in the cost model. Our current implementation does not compress by default but could be extended to do so.
    - *External storage with cryptographic binding*. The raw data could be stored in cheaper external systems (IPFS, decentralised storage or archival services) with only the content-addressed hash anchored on-chain. This would replace the linear storage cost with a constant retrieval cost plus variable external storage pricing.
    - *Differential storage*. Only changes (deltas) between dataset versions are stored rather than complete copies, significantly reducing storage for frequently updated series.
    - *Selective archival*. Only "golden copies" of final published data are stored rather than all intermediate versions, trading off auditability for storage efficiency.

We maintain the baseline assumption of complete dataset storage for three key reasons: (i) **regulatory compliance** often requires maintaining complete, accessible records; (ii) **verification simplicity** ensures that users can verify data without relying on external systems that might have availability or cost unpredictability; and (iii) **audit requirements** in statistical organisations typically mandate complete data preservation for reproducibility. Organisations could explore hybrid models where hot data are stored locally while cold data migrate to cheaper storage tiers with appropriate retrieval guarantees.

- **Merkle proof storage grows logarithmically.** Each dataset's inclusion proof requires $\lceil \log_2(b) \rceil$ hash values, but this cost is partially offset by amortising batch-level storage.
- **Fixed batch overhead is amortised.** Costs such as transaction metadata storage and database indexing for batch relationships are usually divided across all datasets in the batch.

For statistical organisations with existing data dissemination infrastructure, adding blockchain anchoring represents an incremental cost. The storage cost for raw datasets is often already incurred through existing systems; our solution adds only the marginal cost of hash computation and blockchain transactions. This makes adoption more feasible compared with solutions requiring complete data restructuring.

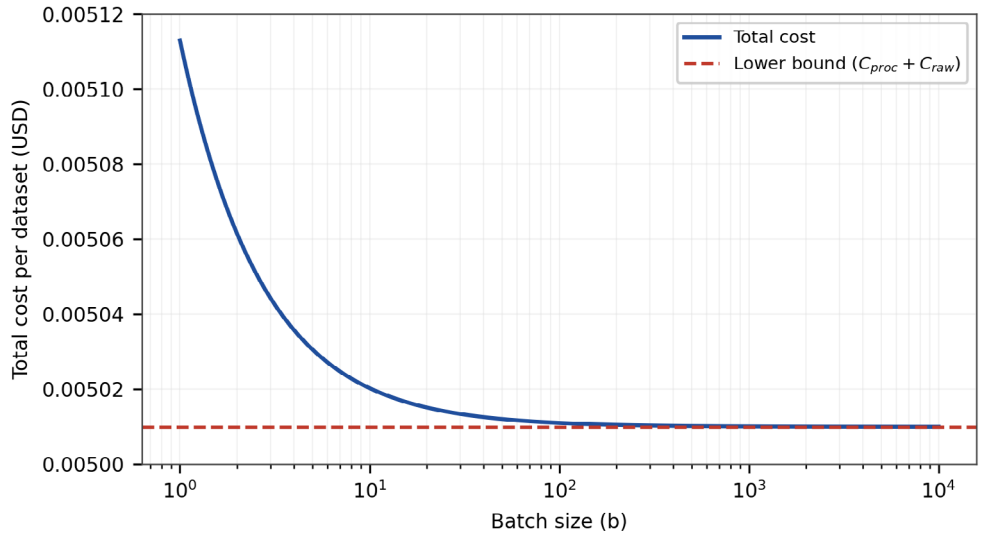
The economic trade-off revealed

This analysis reveals the true economic trade-off in our system. Larger batches significantly reduce the on-chain and fixed overhead costs (both $\propto \frac{1}{b}$), while only

modestly increasing the proof storage cost ($\propto log(b)$). As shown in Graphs 2 and 3, the on-chain and fixed overhead components decrease with batch size, creating clear economies of scale; the proof storage component increases logarithmically, introducing a slight counter-incentive against very large batches. The constant components (raw storage and processing) represent a lower bound that cannot be reduced through batching alone. In most practical scenarios, batching is economically favourable, as it reduces the dominant variable costs by one to two orders of magnitude with minimal impact on latency. All numerical evaluations assume a per-month horizon (T = 1).

Total cost per dataset vs batch size

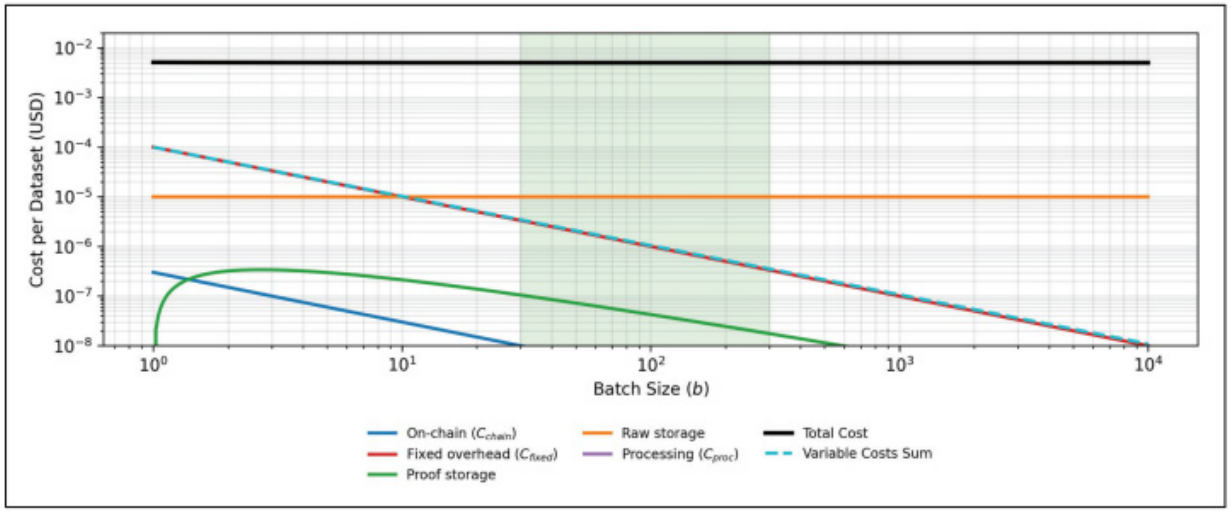
Graph 2



The y-axis is linear and zoomed in to illustrate the ≈2% cost reduction. The cost approaches the theoretical lower bound of $0.00501 (processing plus raw storage) as batch size increases.

Source: Authors

Decomposition of costs Graph 3



The total cost (black line) appears flat because it is dominated by the constant processing cost (purple line, $0.005). The variable costs (cyan dashed) decrease linearly (1/b), illustrating where the efficiency gains come from. The total cost (black curve) represents $C_{total}(b) = f\rho/b + s\sigma + Cfixed/b + c + k \log_2 b\ \sigma_{proof}$, where each dataset incurs its own Merkle proof storage cost $k \log_2 b\ \sigma_{proof}$, which grows logarithmically with batch size. The green region ($30 < b < 300$) represents the economic sweet spot where variable costs are significantly reduced without incurring the latency penalties of very large batches.

Source: Authors

6.2 Stochastic batching, latency cost and optimisation

The deterministic model above ignores the economic cost of waiting (latency) while a dataset waits to be included in a batch. We now extend the model to capture this trade-off: batching lowers chain costs but increases per-dataset delay.

6.2.1 Expected waiting time

Assume datasets arrive according to a Poisson process with rate λ_a- (datasets per second). This memoryless property means that inter-arrival times are exponentially distributed with a mean $\frac{1}{\lambda_a}$. When forming batches of size b, consider the waiting time for a randomly arriving dataset.

1. **Position in batch.** Owing to the memoryless property, a random arrival is equally likely to be in any position k within the batch, where k = 1, 2, …, b.
2. **Waiting time by position.** If a dataset arrives as the k-th element in a batch, it must wait for (b − k) more datasets to arrive before anchoring.
3. **Expected waiting time.** The expected waiting time for position k is $\frac{b-k}{\lambda_a}$, since the expected time between arrivals is $\frac{1}{\lambda_a}$.

4. **Average over positions.** The overall expected waiting time is the average over all positions:

$$E[wait|b] = \frac{1}{b}\sum_{k=1}^{b}\frac{b-k}{\lambda_a} = \frac{(b-1)}{2\lambda_a}$$

5. **Large batch approximation.** For b ≫ 1, we have $E[wait|b] \approx \frac{b}{2\lambda_a}$.

The factor of $\frac{1}{2}$ appears because, on average, a random arrival occurs halfway through the batch formation process.

6.2.2 Complete cost model with delay

Introduce was the marginal economic cost of waiting per dataset per second (USD/s). The expected delay cost per dataset is:

$$C_{delay}(b) \approx w \cdot \frac{b}{2\lambda_a}$$

Including delay, the per-dataset cost becomes:

$$C_{tot\_delay}(b,T) = \frac{f}{b}\cdot\rho + T\cdot\left[s\cdot\bar{\sigma} + \frac{k}{2^{20}}\cdot\lceil\log_2 b\rceil\cdot\sigma_{proof}\right] + \frac{C_{batch\_fixed}}{b} + c + w\cdot\frac{b}{2\lambda_a}$$

This function has the familiar convex trade-off between the decreasing first term and the increasing last term. The optimal batch size b* minimises this total cost.

6.2.3 Optimal batch size

Treating b as continuous, restricting to b ≥ 1 (the feasible integer domain after clamping), relaxing $\lceil\log_2 b\rceil$ to $\log_2 b = \ln b / \ln 2$, and differentiating:

$$\frac{dC_{tot_{delay}}(b,T)}{db} = -\frac{f}{b^2}\cdot\rho + \frac{T\sigma_{proof}}{2^{20}b\ln 2} + \frac{C_{batch\_fixed}}{b^2} + c + \frac{w}{2\lambda_a}$$

where we used $\frac{d}{db}\left[\frac{k}{2^{20}}\sigma_{proof}\log_2 b\right] = \frac{k\sigma_{proof}}{2^{20}\ln 2}$. Setting the derivative to zero and multiplying by b^2 yields the quadratic first-order condition

$$\frac{w}{2\lambda_a}b^2 + T\frac{k\sigma_{proof}}{2^{20}\ln 2}b = f\rho + C_{batch\_fixed}$$

This quadratic $Ab^2 + Bb - C = 0$, with $A = \frac{w}{2\lambda_a}$, $B = T\frac{k\sigma_{proof}}{2^{20}\ln 2}$ and $C = f\rho + C_{batch\_fixed}$, admits the exact positive root

$$b^*_{exact} = \frac{\left(-B + \sqrt{B^2 + 4AC}\right)}{2A}$$

which is well defined since all parameters are positive. In the regime of the numerical examples that follow (Section 6.5), on-chain fees and waiting costs dominate; the proof-storage term B is several orders of magnitude smaller than the other coefficients. Dropping B (equivalently, sending B → 0) yields the simplified closed form

$$b^* \approx \sqrt{\frac{2\left(f\rho + C_{batch_{fixed}}\right)\lambda_a}{w}}$$

which we use in the multi-class extension of Section 6.3. When $C_{batch\_fixed} \ll f\rho$ the formula reduces to $b^* \approx \sqrt{(2f\rho\lambda_a/w)}$, but the parameter values in the numerical examples below place us outside that regime, so the full form is retained for quantitative work.

Interpretation:

- If on-chain fees fρ are large relative to the value of delay (ie w small), b* grows and batching is beneficial.
- If delay costs w dominate, b* < 1 and the model recommends per-dataset anchoring (or very small batches).
- The fixed overhead term introduces an additional incentive for batching owing to amortisation, while the proof storage term ($\propto \log(b)$) introduces a slight counter-incentive against very large batches.

6.3 Multi-class batching optimisation

Real-world SDMx data have varying urgency. We extend the model to support K priority classes with different delay costs w1 > w2 > ⋯ > wK. (To avoid clashing with the proof-node-size parameter k used in Section 6, the class index in this subsection is denoted ℓ.)

Multi-class cost function. For class ℓ with arrival rate $\lambda_{a,\ell}$ and batch size b_ℓ, the per-class contribution to the rate cost (including the amortised batch overhead) is

$$C^{\text{rate}}_{(\text{ell})}(b_{\text{ell}}) = \lambda_{a,\text{ell}}\left[\frac{f\rho}{b_{\text{ell}}} + \frac{C_{\text{batch\_fixed}}}{b_{\text{ell}}} + c + \frac{w_{\text{ell}} \cdot b_{\text{ell}}}{2\lambda_{a,\text{ell}}}\right].$$

The total rate cost is $C_{rate} = \Sigma_{\ell=1}^{K} C^{(\ell)}_{rate}(b\_\ell)$. Because raw dataset storage is constant in b_ℓ and Merkle proof storage is only logarithmic (log b_ℓ) and numerically negligible in the XRPL regime (see Section 6.2.3), they are omitted from the marginal rate optimisation above.

Optimal multi-class batching. Differentiating $C^{(\ell)}_{rate}$ with respect to b_ℓ and setting to zero yields

$$b^*_{\text{ell}} = \sqrt{\frac{2\left(f\rho + C_{\text{batch\_fixed}}\right)\lambda_{a,\text{ell}}}{w_{\text{ell}}}}, \text{ for ell} = 1, \dots, K,$$

which reduces to the single-class formula $b^* = \sqrt{(2(f\rho + C_{batch\_fixed})\lambda_a/w)}$ when K = 1. This model assumes that each priority class is anchored in a separate XRPL transaction to maintain latency independence; mixed-class batching within a single transaction would share the fee but break the independent latency optimisation. Higher priority classes (larger w_ℓ) use smaller batches, providing lower latency at higher per-dataset cost; busier classes (larger $\lambda_{a,\ell}$) tolerate larger batches because their waiting cost is amortised across more arrivals (Graphs 4 and 5).

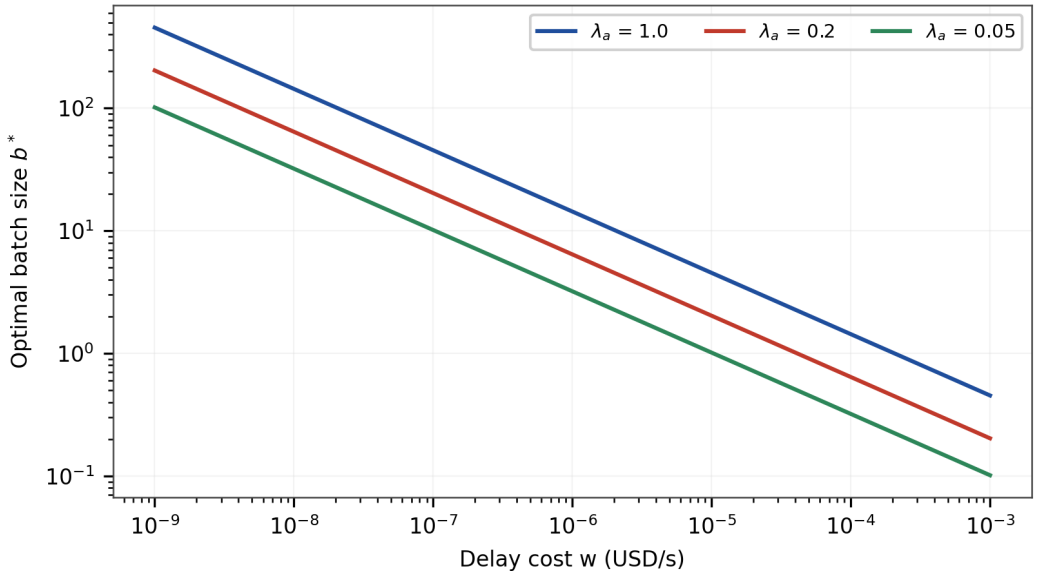
6.3.1 Numerical examples (batching optimisation)

Using the above closed-form approximation, we show a few numerical examples that illustrate some regimes of interest. We use the XRPL network minimum fee and a representative conversion rate: f = 10^{-5} XRP (ie 10 drops, the network base fee) and

ρ = 0.30 USD/XRP, so fρ = 3 × 10^{-6} USD, and the fixed batch overhead C_{batch_fixed} = 10^{-4} USD. Because C_{batch_fixed} exceeds fρ by a factor of approximately 33, the corrected formula that retains the overhead term is used:

$$b^* = \sqrt{\frac{2(f\rho + C_{batch\_fixed})\lambda_a}{w}}$$

Delay cost and optimal batch size, by dataset arrival rate — Graph 4



Lower delay costs favour larger batches, while high-value data requiring immediate anchoring favour smaller batches.

Source: Authors

Example cases

- **Case A:** λ_a = 0.5 datasets/s (one dataset every 2 seconds), w = 10^{-3} USD/s (delay costs $0.001 per second). Taking T = 1 month as the evaluation horizon:

  $$b^* \approx \sqrt{(2 \times (3\ 10^{-6} + 10^{-4}) \times 0.5 / 10^{-3})} = \sqrt{(1.03\ 10^{-1})} \approx 0.321.$$

  Interpretation: since $b^* < 1$, the economic cost of delay dominates and frequent or single-document anchoring is preferred (given these parameters). Because the batch size is a positive integer, the unconstrained $b^* \approx 0.321$ is clamped to the feasible domain, giving the operating point $\max(1, \text{round}(b^*)) = 1$ (per-dataset anchoring). For non-integer b^*, rounding rather than taking the ceiling avoids upward bias at the continuous-to-discrete conversion.

- **Case B:** λ_a = 0.1 datasets/s, w = 10^{-7} USD/s (extremely low delay cost):

  $$b^* \approx \sqrt{(2 \times (3 \cdot 10^{-6} + 10^{-4}) \times 0.1 / 10^{-7})} = \sqrt{206} \approx 14.35,$$

  so modest batching becomes worthwhile.

- **Case C:** for a very small delay cost w = 10^{-9} USD/s and λ_a = 1.0:

$$b^* \approx \sqrt{(2 \times (3 \cdot 10^{-6} + 10^{-4}) \times 1.0 / 10^{-9})} = \sqrt{(2.06 \cdot 10^5)} \approx 453.9,$$

ie large batching becomes economically sensible.

Notes on approximations

- The waiting-time formula $(b - 1)/(2\lambda a)$ is an approximation assuming uniformity across the filling interval; exact waiting-time distributions may require more detailed queuing analysis (see, for example, M/G/1 batch-accumulation models) (Gross and Harris, 1998).
- The model ignores discrete constraints (maximum memo size, proof sizes, operational caps) and assumes that a cost w can be specified – in practice w may be derived from SLA penalties, reputational weights, or a shadow price estimated by management or other factors.
- The closed-form solution here simply gives intuition and a tractable, gentle starting point for policy. For practical deployments, we recommend running sensitivity analysis across realistic arrival rates and delay valuations.

6.4 Batching latency optimisation (deterministic model)

Anchoring frequency trades off cost per dataset against latency. Let b be batch size and $\tau(b)$ the anchoring latency (time between dataset publication and root anchoring). Suppose $\tau(b) = \tau_0 + \alpha b$ captures a simple queuing/aggregation latency, where τ_0 is fixed overhead and α the per-dataset aggregation delay. Consider minimising expected cost subject to a maximum allowed latency $\tau(b) \leq \tau_{max}$. The constrained optimisation is:

$$\min_b C_{\text{total}}(b, T) = \frac{f\rho}{b} + T\left[s\bar{\sigma} + \frac{k}{2^{20}}\lceil \log_2 b \rceil \sigma_{\text{proof}}\right] + \frac{C_{\text{batch\_fixed}}}{b} + c,$$

$$\text{s.t. } b \leq \frac{\tau_{\max} - \tau_0}{\alpha} =: b_{\max},$$

where $T > 0$ is the evaluation horizon in months (set to $T = 1$ in the numerical examples below), so $C_{total}(b,T)$ is the total expected cost per dataset over that horizon. The storage and proof terms are multiplied by T because they accrue each month; the one-time anchor and batch-fixed costs ($f\rho/b$, C_{batch_fixed}/b, c) are incurred once per dataset regardless of the horizon.

If $b_{max} \geq 1$, the unconstrained minimiser of $C_{total}(b)$ over real $b > 0$ is obtained by setting the derivative to zero. Multiplying by b^2 yields a linear equation in b, giving the exact closed-form minimiser $b^* = (f\rho + C_{batch\_fixed})\, 2^{20} \ln 2 / (T\, k\, \sigma_{proof})$ when the delay-cost term is omitted (recalling $T = 1$ in the numerical examples below). When the delay-cost term of Section 6.2.3 is included, the first-order condition becomes a quadratic equation, which also admits an exact algebraic solution. If the unconstrained optimum b^* exceeds b_{max}, the optimal constrained solution is b_{max}; if $b^* < 1$, the optimum on the feasible domain $[1, b_{max}]$ is 1. The global optimum is therefore $\max(1, \min(b^*, b_{max}))$, requiring no numerical search in any regime.

6.5 Numerical examples

Assume: f = 10^{-5} XRP (10 drops, the XRPL base fee); conversion ρ = 0.30 USD/XRP (example); dataset size s = 0.5 MB; storage price $\bar{\sigma}$ = 0.00002 USD/MB-month (amortised per dataset); proof node size k = 64 bytes; proof storage cost σ_{proof} = 0.00001 USD/MB-month; fixed batch overhead C_{batch_fixed} = 0.0001 USD; and processing cost c = 0.005 USD.

Case A: batching b = 1,000.

$$C_{\text{chain}} = \frac{10^{-5}}{10^{3}} \cdot 0.30 = 3 \times 10^{-9} \text{ USD},$$

$$C_{\text{store}} = 0.5 \times 0.00002 + 64 \cdot \lceil \log_2 1000 \rceil \cdot \frac{0.00001}{2^{20}} + \frac{0.0001}{1000}$$
$$= 0.00001 + 6.1 \times 10^{-9} + 1 \times 10^{-7} \text{ USD},$$

$$C_{\text{total}} = c + C_{\text{store}} + C_{\text{chain}} \approx 0.0050101091 \text{ USD}.$$

Case B: large batch b = 10^6.

$$C_{\text{chain}} = \frac{10^{-5}}{10^{6}} \cdot 0.30 = 3 \times 10^{-12} \text{ USD},$$

$$C_{\text{store}} = 0.5 \times 0.00002 + 64 \cdot \lceil \log_2 10^6 \rceil \cdot \frac{0.00001}{2^{20}} + \frac{0.0001}{10^{6}}$$
$$= 0.00001 + 1.22 \times 10^{-8} + 1 \times 10^{-10} \text{ USD},$$

$$C_{\text{total}} = c + C_{\text{store}} + C_{\text{chain}} \approx 0.0050100122 \text{ USD}.$$

Case C: per-dataset anchoring b = 1.

$$C_{\text{chain}} = 10^{-5} \cdot 0.30 = 3 \times 10^{-6} \text{ USD},$$

$$C_{\text{store}} = 0.5 \times 0.00002 + 0 + \frac{0.0001}{1} = 0.00001 + 0 + 0.0001 \text{ USD},$$

$$C_{\text{total}} = c + C_{\text{store}} + C_{\text{chain}} \approx 0.005113 \text{ USD}.$$

Interpretation: with reasonable batching, the dominant costs are processing and raw storage; on-chain fees and proof storage are negligible at XRPL rates in these examples. The fixed batch overhead becomes significant only for very small batches. The results indicate that with reasonable batching (eg b > 50), the marginal cost of blockchain anchoring and proof storage becomes negligible compared with the constant costs of processing and raw data storage. Consequently, the dominant economic factor becomes the fixed cost of verification and storage (C_{proc} + $s\bar{\sigma}$), rendering the system highly scalable. The 1/b decay of the variable components implies that even for high-frequency data, modest batches yield significant savings, while the penalty for very small batches is bounded by the low nominal fees of the XRPL.

Section summary. The cost model decomposes per-dataset cost into on-chain, storage, processing and delay components, and admits a closed-form economically optimal batch size $b^* = \sqrt{(2(f\rho + C_{batch\_fixed})\lambda_a/w)}$ for a single class, with a per-class extension for mixed urgencies. In the XRPL regime, on-chain fees are negligible at any reasonable batch size; the operating point is set by waiting cost and raw storage.

7. Performance and evaluation

7.1 Test environment and methodology

All numbers reported below were obtained on the proof-of-concept deployment described in Section 3 and should be read as prototype-level rather than production-grade. The setup was as follows.

- **Hardware and topology:** a single developer workstation running the open-source docker-compose stack (Rusev, 2026), with the FastAPI backend, Streamlit frontend[7], PostgreSQL database and Keycloak identity provider each in its own container on the same host. No tuning beyond the defaults shipped in the repository was applied.
- **Network:** a standard internet uplink to a public XRPL JSON-RPC endpoint. In our regime, end-to-end latency is dominated by the ledger-close cadence rather than by network round-trip time, so we do not report a separate network round-trip figure.
- **Blockchain:** the XRPL DevNet (`s.devnet.rippletest.net`) for all reported runs, as configured in the open-source reference implementation. DevNet uses the same transaction format and ledger-close cadence as mainnet, so per-transaction latency is representative; absolute fees on mainnet differ but remain in the sub-cent range.[8]
- **Dataset corpus:** a synthetic SDMX-ML 2.1 corpus generated from BIS public structure definitions, spanning file sizes from 10 KB (a few series) to 8 MB (several thousand series). For batched runs, batches were drawn uniformly at random from this corpus.
- **Methodology:** each reported latency is the median of repeated runs against the DevNet endpoint after a short warm-up to amortise connection setup; throughput is characterised by sustained submission with a configurable number of concurrent producer threads. Confirmation latency is measured from transaction submission until the transaction first appears in a validated ledger.
- **What is not measured:** we did not run the system at sustained mainnet load, behind an enterprise web application firewall, with HSM-backed signing in line, or under adversarial conditions. The numbers therefore characterise the prototype's efficiency, not its production resilience.

7.2 Empirical measurements

Empirical measurements from our prototype show:

- **Publication time** (canonicalisation, hashing, aggregation and off-chain bookkeeping): 3–5 seconds (median); anchoring confirmation follows XRPL finality.

[7] Streamlit Docs: "Deploy Streamlit using Docker", https://docs.streamlit.io/deploy/tutorials/docker.

[8] XRPL.org: "XRP Ledger documentation and developer resources", https://xrpl.org/docs.

- **Verification time** (canonicalisation, hashing, on-chain query or local database indexed lookup of anchor metadata): 1–2 seconds for individual datasets.
- **Verification scalability:** end-to-end verification time scales as O(|X| + log n), dominated by the linear parsing and canonicalisation of the SDMx payload X; the cryptographic Merkle proof verification adds only O(log n) overhead. For the prototype corpus (files up to 8 MB), the total remained under 2 seconds.
- **Throughput:** because Merkle batching amortises the single anchoring transaction across all leaves in a batch, system-level throughput is bounded primarily by the chosen batch cadence and the XRPL ledger-close cadence rather than by per-dataset cryptography. We did not run a sustained saturation benchmark on the prototype; for any concrete deployment, throughput at the chosen operating point should be measured against the publisher's target load.
- **Storage overhead:** approximately 1–2 KB per dataset for hash/proof metadata, depending on proof encoding and batch size. The Merkle proof size grows logarithmically with batch size, with each additional level adding 64 bytes per dataset (SHA3-512).
- **Availability:** the prototype runs on a single host and was not measured against a service-level commitment. Production availability is dominated by the operational practices of the deploying organisation (redundancy, monitoring, on-call response) rather than by the anchoring software itself, so we deliberately abstain from quoting an uptime figure for the prototype.

Caveats on production extrapolation. The figures above were obtained under the conditions of Section 7.1. We deliberately avoid the term "production-ready" for the prototype as such: real deployments will need an HSM-backed signing path, validated XRPL node pinning, formal change management, an incident-response playbook, and load testing at the publishing institution's expected peak rate. The cryptographic core (canonicalisation, hashing, Merkle aggregation, anchoring, verification) is what we claim to be production-oriented; the surrounding operational envelope is the responsibility of the deploying organisation.

Comparison with alternatives. Table 2 compares representative approaches. The table is intentionally compact for at-a-glance reading; the per-cell methodology behind each value is given in the paragraphs that follow, so that readers can judge whether the comparison is fair across these technologies.

Comparison with alternative systems — Table 2

	SDMx semantics	Merkle batching	VC binding	Cost model
Verification time	≈1–2 s	Minutes–hours	Minutes–hours	Milliseconds (OCSP)
Transaction fee	≈$0.000003	≈$0.50[4]	≈$5.00 (L1); <$0.01 (L2)	$0.00 (amortised)
Integration complexity	Low (API-based)	Medium	High	Low
Real-time verification	Yes	Limited	Limited	Yes
SDMx compliance	Full	Variable	Variable	Variable
Scalability	High[1]	Medium	Low	High[2]
Energy efficiency	High	Medium	High[3]	High

[1] XRPL's high scalability refers to its theoretical decentralised throughput capacity (~1,500 transactions per second) and the system's ability to batch thousands of datasets into a single on-chain transaction. [2] Scalability for PKI is rated high owing to the elastic, centralised infrastructure of OCSP responders and content delivery networks; however, this relies on the certification authority's operational capacity and trust model. [3] Ethereum transitioned to proof-of-stake (the Merge, September 2022), reducing energy consumption by ~99.95% relative to proof-of-work. Its current energy profile (~0.03 TWh/yr) is comparable to a global server network and is rated high accordingly. The $5.00 transaction fee is for Ethereum L1 and is volatile; L2 rollups (eg Arbitrum, Optimism) offer fees of fractions of a cent. [4] Chainlink Data Feeds charge subscription-based access rather than per-query fees; the $0.50 figure shown is a representative on-chain gas cost to consume a feed update, not a Chainlink service fee.

Sources: Authors

How each cell was computed

- **Verification time.** For our system, the value is the end-to-end client-side verification latency measured under Section 7.1 (canonicalisation plus hashing plus on-chain lookup against a public XRPL node plus Merkle recomputation). For Ethereum, "minutes–hours" reflects typical block finality of ≈12 seconds (the post-Merge slot time) plus the practical recommendation of waiting 12–32 confirmations (Zheng et al, 2017) before treating data as final, plus the round trip to a feed contract. For Chainlink, the figure reflects the documented update cadence of standard data feeds (heartbeat windows ranging from several minutes to multiple hours) plus on-chain read latency. For traditional PKI, "milliseconds" reflects OCSP stapling, which provides near-instant verification; the key PKI limitation is centralised trust in the certification authority rather than verification speed.
- **Transaction fee.** XRPL: median observed network fee of 10 drops (10^{-5} XRP) at $\rho \approx 0.30$ USD/XRP per anchoring transaction (per batch, not per dataset). For a typical batch of $b = 1{,}000$ datasets the fee per dataset would be 3×10^{-9} USD, but the table reports the per-transaction cost for direct comparison with other systems that charge per operation. Ethereum: a representative mainnet gas cost for a small `SSTORE`-bearing transaction at a 20 gwei gas price and ETH at around USD 2,500; we acknowledge that gas prices are volatile. Chainlink: a representative cost reported by feed operators for a single on-chain feed update including LINK gas. Traditional PKI: a representative public certification authority certificate / signing fee for an organisational identity certificate (a one-time or annual cost amortised over an unlimited number of datasets).

- **Integration complexity.** A qualitative ranking based on the number of new components a statistical authority must operate to use the system: an HTTPS API client (low) versus a custom oracle node and smart contract integration (medium) versus a full smart contract codebase or PKI hierarchy with HSMs, registration authorities and certificate revocation list infrastructure (high).
- **Real-time verification.** "Yes" indicates that finality or verification is reached within the order of seconds and the verification client can be invoked synchronously from an interactive user interface (traditional PKI achieves this via OCSP stapling, though it relies on centralised trust in the certification authority); "limited" and "no" reflect the latencies above.
- **SDMx compliance.** "Full" means that the system natively understands SDMX-ML structure and per-`<Series>` granularity; "variable" means that SDMx support is possible but must be built on top of the technology.
- **Scalability.** An order-of-magnitude characterisation based on the underlying ledger's documented throughput (XRPL: ∼1,500 transactions per second[9]; Ethereum L1: ∼15 transactions per second; Chainlink: bounded by update cadence and oracle node throughput; PKI: bounded by certification authority / OCSP responder capacity).
- **Energy efficiency.** Reflects the consensus mechanism: XRPL's federated Byzantine agreement is approximately energy-neutral relative to a standard server; Ethereum post-Merge is much improved over proof-of-work but still requires a global validator set; Chainlink inherits the energy profile of its target chain; PKI is essentially server-bound.

The values are best read as the technology's typical operating point for the SDMx-anchoring use case considered here. A different use case (eg high-value financial settlement on Ethereum) would weight these dimensions differently.

Discussion of comparison parameters

The evaluation criteria in Table 2 were carefully selected to highlight the distinctive advantages of our implementation for SDMx data dissemination. Each parameter was assessed based on both technical capabilities and practical considerations for statistical organisations.

Integration complexity. Our system achieves low integration complexity through its API-based design and Docker containerisation, which allows seamless integration with existing SDMx workflows. This contrasts with medium complexity for Chainlink, which requires custom oracle development, and high complexity for Ethereum, which needs specialised smart contract development. Traditional PKI is also rated low because standard PKI (eg XML digital signatures) is natively supported by virtually all enterprise XML parsers; however, managing revocation and certification authority trust hierarchies introduces operational overhead not captured by this rating.

Real-time verification. The "yes" rating for our implementation reflects XRPL's 3–5 second finality and our optimised verification algorithms. While for some use

[9] XRPL.org: "XRP Ledger documentation and developer resources", https://xrpl.org/docs.

cases a few seconds may seem negligible, several critical applications demand near-instant verification:

- **Automated AI agents and trading systems.** AI agents making real-time decisions require immediate data integrity verification. For GDP data or other key economic indicators, a delay of minutes or hours could mean millions in trading losses or missed opportunities in automated trading systems that react within milliseconds to economic announcements.
- **High-frequency financial instruments.** Derivatives, perpetual futures and inflation-linked products often have trigger mechanisms that depend on timely data verification. A smart contract for an inflation-linked bond, for example, needs immediate access to verified inflation data for interest rate calculations and automated coupon payments.
- **Regulatory compliance and audit trails.** Real-time verification enables continuous compliance monitoring for financial institutions that must demonstrate that they are using authoritative data for risk calculations and regulatory reporting.
- **Nowcasting and economic forecasting.** Economic nowcasting models that incorporate high-frequency indicators require timely data verification to produce accurate real-time estimates for monetary policy decisions.

Chainlink and Ethereum offer only limited real-time capabilities owing to longer block times (minutes for Ethereum) and oracle reporting intervals (often hours for Chainlink data feeds).

Ecosystem synergy. Beyond individual feature comparisons, the most significant advantage emerges when trustworthy data are available on the same blockchain that hosts digital assets (central bank digital currencies, stablecoins, tokenised deposits, derivatives). This creates a synergistic ecosystem in which:

- **Programmable trust.** Smart contracts can programmatically verify data integrity before execution, enabling automated, regulatory-compliant financial instruments that are trust-minimised and auditable.
- **Reduced counterparty risk.** When data, assets and execution logic reside on the same chain, there is no need to trust cross-chain bridges or external data feeds, reducing systemic risk.
- **Innovation acceleration.** Developers can build novel financial products that combine verifiable economic data with programmable money, creating opportunities for automated monetary policy implementation, transparent inflation-indexed instruments and dynamic risk management tools.

Our chain-agnostic architecture recognises that the market will probably converge on a handful of dominant chains for digital assets. The system can be deployed across multiple chains simultaneously, ensuring that verifiable data are available wherever digital assets are issued and traded. This multi-chain strategy also enhances redundancy, trust distribution and interoperability across different blockchain ecosystems.

SDMx compliance. Our "full" compliance rating stems from the native support for SDMx standards throughout our architecture, including SDMX-ML processing, metadata handling and easy extension to registry integration.

Scalability. The high scalability of our implementation results from the efficient Merkle tree batching approach and XRPL's proven capability to handle 1,500+ transactions per second. Traditional PKI is also rated high owing to the elastic capacity of centralised OCSP responders and content delivery networks, though this relies on the certification authority's operational capacity and trust model rather than decentralised throughput.

Energy efficiency. Our high energy efficiency rating derives from XRPL's federated Byzantine agreement consensus, which avoids both the mining workload of proof-of-work and the staking/attestation overhead of proof-of-stake; per-transaction energy is dominated by ordinary server CPU usage at each validator. Ethereum is also rated high post-Merge (see the notes to Table 2). The ratings for both systems reflect modern, energy-efficient consensus: XRPL (federated Byzantine agreement) and Ethereum (proof-of-stake) are both orders of magnitude more efficient than legacy proof-of-work systems.

Section summary. On the prototype, publication takes 3–5 seconds and verification 1–2 seconds, with verification scaling as $O(|X| + \log n)$ – linear in payload size plus logarithmic cryptographic overhead. The numbers were obtained on a single-host XRPL DevNet deployment with synthetic SDMx corpora; they characterise the prototype's efficiency, not a fully hardened production deployment. The comparison with Chainlink, Ethereum and PKI is documented cell by cell so that the reader can re-weight the table for a different use case.

8. Use cases and econometric checks

8.1 Immutable version authentication

Each dataset release is hashed and either individually anchored or included via a Merkle proof in a batch root R anchored on XRPL. Publisher signatures (X.509 or DID) could optionally add identity binding. Verifiers retrieve (h, π, R, tx) and check root inclusion and signature validity.

8.2 Redistribution of trust framework

Publishers may issue W3C Verifiable Credentials that bind dataset identifiers and fingerprints h (Sporny et al, 2018). Re-publishers bundle dataset plus credential plus inclusion proof; consumers verify credential signatures and Merkle inclusion locally. For selective disclosure, BBS+ or other selective-disclosure schemes allow revealing only required fields.

8.3 Automated monitoring and anomaly detection

Anchored metadata enable automatic monitoring of time series releases. This can be generalised for any data point (not necessarily sequential or time series), such as revisions across measures. Let y_t be a reported series (eg monthly CPI). Define the revision or change $d_t = y_t - y_{t-1}$. A simple cumulative sum (CUSUM) detector can

signal abnormal revisions that may indicate tampering or publication errors (Page, 1954): define

$$S_0 = 0, S_t = \max\{0, S_{t-1} + (d_t - \mu) - k\}$$

pawhere μ is the expected mean change and k a slack parameter; an alarm is raised if $S_t > h$. (Within this subsection only, k and h denote the CUSUM slack and alarm threshold respectively, and are distinct from the Merkle proof-node size k of Section 6 and the fingerprint h of Section 4.) The anchored hashes and timestamps make it trivial to map flagged releases to specific published payloads and inclusion proofs for forensic analysis.

8.3.1 Recommended extended econometric checks

In addition to CUSUM-style sequential detectors, we recommend combining:

- CUSUM and CUSUM-of-squares for small but persistent deviations in revisions (Page, 1954; Bai and Perron, (1998);
- sequential change-point methods and likelihood-ratio style detectors from Basseville and Nikiforov (1993) for online detection of abrupt tampering events;
- structural-break tests (Bai and Perron) to detect breaks in regression relationships or in the leading indicators that predict series behaviour (Bai and Perron 1998); and
- robust time series diagnostics such as tests for heteroskedasticity, outlier detection and seasonal adjustment validation (eg comparison with pre-anchored benchmark releases).

8.3.2 Simple numerical example (CUSUM)

Suppose a monthly series of revisions d_t (in basis points) over eight months is d = [0.1, 0.05, −0.02, 0.12, 0.15, 0.14, 0.13, 1.20]. Let the expected mean revision μ = 0.08 and slack k = 0.02. Computing S_t:

$$\begin{aligned}
S_1 &= \max(0, 0 + (0.10 - 0.08) - 0.02) = 0.00,\\
S_2 &= \max(0, 0.00 + (0.05 - 0.08) - 0.02) = 0.00,\\
S_3 &= \max(0, 0.00 + (-0.02 - 0.08) - 0.02) = 0.00,\\
S_4 &= \max(0, 0.00 + (0.12 - 0.08) - 0.02) = 0.02,\\
S_5 &= \max(0, 0.02 + (0.15 - 0.08) - 0.02) = 0.07,\\
S_6 &= \max(0, 0.07 + (0.14 - 0.08) - 0.02) = 0.11,\\
S_7 &= \max(0, 0.11 + (0.13 - 0.08) - 0.02) = 0.14,\\
S_8 &= \max(0, 0.14 + (1.20 - 0.08) - 0.02) = 1.24,
\end{aligned}$$

If the alarm threshold is h = 0.5, an alarm would trigger at t = 8. Anchored proofs allow immediate retrieval of the exact SDMx payload corresponding to t = 8 and verification of the canonical hash and inclusion proof for forensic review.

8.4 Nowcasting with verifiable input data

Nowcasting models that predict current economic conditions in real time rely heavily on the integrity of high-frequency input data. Our system provides crucial trust guarantees for nowcasting pipelines by ensuring the authenticity of input datasets.

Verifiable nowcasting pipeline. A typical nowcasting workflow could be enhanced as follows.

1. **Input verification.** Each high-frequency indicator (eg surveys, retail sales, industrial production) is verified against its blockchain anchor before being fed into the nowcasting model.
2. **Model integrity.** The nowcast result can itself be anchored, creating an immutable record of the prediction and its input data provenance.
3. **Revision tracking.** As preliminary data are revised, the nowcast can be recomputed and the revisions tracked via the immutable audit trail.

Example: GDP nowcasting with verified inputs. Consider a mixed-frequency nowcasting model (Mariano and Murasawa, 2003) for quarterly GDP growth:

$$y_t^Q = \alpha + \sum_{j=1}^{p} \beta_j \, y_{t-j}^Q + \sum_{k=1}^{K} \gamma_k \sum_{i=0}^{2} x_{k,3t-i}^M + \varepsilon_t,$$

where y^Q_t is quarterly GDP growth and $x^M_{k,t}$ are monthly indicators. Each $x^M_{k,t}$ can be verified against its blockchain commitment before model estimation. If verified data integrity reduces input errors, this could translate into improved nowcast accuracy. The resulting gains for monetary policy decisions are discussed in Giannone et al (2008) and Bańbura et al (2010).

Empirical benefits for nowcasting.

- **Reduced model risk:** eliminates the risk of nowcasts based on tampered or corrupted input data.
- **Auditable decisions:** central banks can demonstrate the exact data inputs used for policy decisions.
- **Improved revision analysis:** clear tracking of how nowcasts change as data revisions occur.
- **AI agent trust:** automated nowcasting systems can cryptographically verify data inputs before processing.

Implementation for real-time nowcasting. The system could support near real-time nowcasting workflows through:

- API endpoints for automated verification of incoming data streams;
- webhook notifications when new verified data are published, as an extension;
- agnostic integration with popular nowcasting platforms; and
- support for high-frequency data (daily, weekly indicators) with efficient batching.

Section summary. Anchored fingerprints unlock three classes of use case: tamper-evident version authentication, verifiable redistribution via signed credentials, and automated econometric monitoring (CUSUM, change-point, structural break) tied directly to specific anchored payloads. Nowcasting pipelines benefit because every input series can be verified before it is fed into the model.

9. Formal foundations

The construction of Section 4 and the implementation of Section 5 have so far been justified operationally. This section adds the formal layer: a computational threat model (with Dolev–Yao network capabilities), an explicit single-compromise resilience table, a reduction of the Merkle binding to SHA3-512 second-preimage resistance, an information-theoretic lower bound that the chosen anchor size matches up to a small constant, a zero-knowledge selective-disclosure extension via BBS+ signatures, a multi-publisher accumulator with $O(\log n_p + \log P)$ inclusion proofs, and a mechanism-design analysis of strategic batching across publishers. To keep notation distinct from Section 6, we use κ for the cryptographic security parameter (the currency conversion of Section 6 retains its symbol ρ), and we let λ_j denote the per-publisher Poisson arrival rate that extends λ_a of Section 6 to a P-publisher setting.

9.1 Threat model

We adopt a computational network adversary A with Dolev–Yao capabilities and the limits below.

Capabilities. A may read, drop, reorder, replay and inject arbitrary messages on the public network between any pair of honest parties; submit polynomially many adaptive queries to the hash and signature oracles; and compromise at most one of the following in a single execution: (a) the publisher's backend, observed after the anchoring transaction has been validated; (b) a strict minority of XRPL validators within a publisher's unique node list (UNL); or (c) any consumer's local cache.

Limits. A is computationally bounded with security parameter κ; it cannot break SHA3-512 second-preimage resistance, collision resistance, or Ed25519 EUF-CMA except with negligible advantage in κ, and it cannot violate XRPL's honest-UNL-majority assumption.

Security goals. For a published dataset X with anchor receipt (R, `tx_hash`, VC) we require:

- **INT – integrity:** A cannot produce $X' \neq X$ that verifies against R.
- **ORG – origin authenticity:** A cannot produce a verifying receipt naming a publisher that did not anchor X.
- **TS – timestamp:** A cannot predate the ledger position of `tx_hash`.
- **NR – non-repudiation:** the publisher cannot later disclaim a receipt that verifies against the on-chain anchoring key it controlled at the time.
- **PV – public verifiability:** any consumer can check INT, ORG and TS from the file plus a single ledger lookup.

Table 3 lists which goals survive each isolated compromise.

Properties preserved under each isolated compromise — Table 2

Compromise	INT	ORG	TS	NR	PV
Network (computational / Dolev–Yao)	Y	Y	Y	Y	Y
Publisher backend (post-anchor)	Y	Y	Y	Y	Y
Single XRPL validator (minority)	Y	Y	Y	Y	Y
Consumer local cache	Y	Y	Y	Y	Y
Stolen VC signing key	Y	◦	Y	◦	Y
Stolen XRPL anchoring key	◦	N	Y	N	Y
SHA3-512 collision found	N	◦	Y	◦	N

Y = preserved; N = broken; ◦ = preserved if and only if the qualifier in the text holds.

Sources: Authors

The final three rows of Table 3 extend beyond the single-compromise capabilities (a)–(c) enumerated above: the two key-theft rows and the SHA3-512 collision row are assumption-violating events rather than single-source compromises, and are included to make the cost of those out-of-model failures explicit.

Theft of the credential signing key does not break INT in the base Ed25519 scheme, because the on-chain root R is committed independently of the credential; it enables forged identity claims only until the key is rotated and the rotation event is anchored. (For the BBS+ selective-disclosure extension of Section 9.4, the INT guarantee under credential-key theft is weaker; see the discussion there. The INT entry in Table 3 therefore applies to the Ed25519 scheme; for BBS+ without condition (ii) of Definition 4, INT is ◦ rather than Y.) Anchoring-key theft does not break TS, because the ledger position of past transactions cannot be retroactively altered. The INT entry for anchoring-key theft is ◦ because the hash-binding of any root anchored before the compromise is untouched – forging it still requires a second preimage of SHA3-512 – whereas a thief holding the anchoring key can post new anchors that bind arbitrary forged payloads to fresh roots. Integrity is therefore preserved relative to a root fixed before the theft, but not for anchors created afterwards.

9.2 Security of the domain-separated Merkle construction

Recall from Section 4.1 the tagged leaf and internal hashes of the Merkle tree (Merkle, 1988):

$$H_{leaf}(h) = H(0x00 \,\|\, h), \quad H_{node}(L,R) = H(0x01 \,\|\, L \,\|\, R),$$

where H = SHA3-512. Because h, L and R are always 64-byte SHA3-512 outputs, their lengths are fixed and the len32(·) prefixes would be redundant; the single-byte domain separators (`0x00` versus `0x01`) are already sufficient to prevent concatenation

ambiguity between leaves and nodes. Leaf vectors of size n are padded to $2^{\lceil \log_2 n \rceil}$ by duplicating the final leaf before pairing (for n = 1, $2^0 = 1$ and no padding is needed; the internal root is $H_{leaf}(h_1)$). Denote by $Root_{internal}(h)$ the internal root computed from leaf vector $h = (h_1, \ldots, h_n)$ via H_{leaf} and H_{node}, and define the final root as

$$R = Root(h) = H(0x02 \,\|\, len32(n) \,\|\, Root_{internal}(h)),$$

where len32(n) denotes the 32-bit big-endian encoding of the integer n (unsigned), and $n = |h|$ is the leaf count. Binding n into the root prevents leaf-duplication ambiguity: n = 3 padded to four leaves (duplicating h_3) and n = 4 with the final leaf matching h_3 produce different roots because n differs, even though $Root_{internal}$ would coincide.

Theorem 1 (binding of the tagged root). Let $h \neq h'$ be two ordered leaf vectors of (possibly different) lengths such that Root(h) = Root(h'). Then a collision in H exists.

Proof. Let $n = |h|$ and $n' = |h'|$. From Root(h) = Root(h') we have $H(0x02 \,\|\, len32(n) \,\|\, Root_{internal}(h)) = H(0x02 \,\|\, len32(n') \,\|\, Root_{internal}(h'))$. If the pre-images differ, a collision in H has been exhibited; by collision resistance they must be equal, hence n = n' and $Root_{internal}(h) = Root_{internal}(h')$. Now proceed by induction on the tree height $t = \lceil \log_2 n \rceil$, with the inductive hypothesis that equality of $Root_{internal}$ on two 2^t-leaf padded vectors forces those padded vectors to be equal element-wise (for n > 1, the 2^t leaves are padded by duplicating the final leaf; for n = 1, t = 0 and no padding is needed). For the base case t = 0 (n = 1), $Root_{internal}(h) = H_{leaf}(h_1)$; equality forces $h_1 = h'_1$ (else a collision in H). For t = 1, the padded tree has exactly two leaves (n = 2, no padding needed): $Root_{internal}(h) = H_{node}(H_{leaf}(h_1), H_{leaf}(h_2))$; equality forces both children to match, giving $h_1 = h'_1$ and $h_2 = h'_2$. For t > 1, split the padded leaf vector at the midpoint into left half L and right half R. The internal root is $Root_{internal}(h) = H_{node}(Root_{internal}(L), Root_{internal}(R))$. Equality of the H_{node} output forces matching inputs (else a collision), so $Root_{internal}(L)$ and $Root_{internal}(R)$ match their counterparts in h'. By the inductive hypothesis the left-half leaf vectors match, then the right-half leaf vectors match. Finally, since n = n' is already fixed by the collision-resistant len32 prefix and the padding is a deterministic function of n (duplicate the final real leaf until 2^t leaves are present), element-wise equality of the two 2^t-leaf padded vectors forces equality of the first n real leaves, ie h = h', completing the proof. □

Corollary 2 (second-preimage resistance of inclusion proofs). Given a valid proof π_i for h_i against root R, no probabilistic polynomial-time adversary can output $h'_i \neq h_i$ together with a proof π'_i such that the verifier accepts (h'_i, π'_i, R), except with advantage at most the second-preimage advantage against H.

The Merkle code path `merkle_root_from_hex_hashes` in the open reference implementation is a byte-for-byte realisation of H_{leaf} and H_{node}; Theorem 1 therefore applies to the executable artefact, not only to a stylised abstraction. Inclusion proofs inherit second-preimage resistance via Corollary 2.

9.3 Tight on-chain footprint bound

Let X denote the universe of admissible SDMx batches, and consider any publicly verifiable anchoring scheme that commits, per batch, a binary string of length ℓ bits from which a verifier later checks inclusion of any $X \in X$.

Lemma 3 (on-chain footprint for hash-based schemes). Let κ be the computational security parameter. Any generic hash-based anchoring scheme satisfying INT against an adversary restricted to q hash queries must commit at least $\ell \geq \kappa + \log_2 q$ bits per batch. For a standard probabilistic polynomial-time adversary with $q = \text{poly}(\kappa)$, this yields $\ell \approx \kappa + O(\log \kappa)$. Against a quantum adversary, Grover's algorithm finds a second preimage in $O(2^{\ell/2})$ quantum queries. To ensure that the best quantum attack requires at least 2^κ quantum operations, we equate $2^{\ell/2} \geq 2^\kappa$, which yields $\ell \geq 2\kappa$. For concrete parameters $\kappa = 256$, we obtain $\ell \geq 512$, which our 512-bit SHA3-512 output satisfies exactly. (Non-hash-based schemes, eg lattice-based or algebraic commitments, may have different security bounds not captured by this lemma.)

Proof sketch. The commitment induces a function $f_{com}: X \to \{0,1\}^\ell$. In our threat model (Section 9.1), the publisher is honest at anchoring time and the adversary only compromises the backend post-anchor or acts as a network adversary. An INT break therefore requires an adversary, given $R = f_{com}(X)$ for an honestly generated X, to find $X' \neq X$ such that $f_{com}(X') = R$. This is a second-preimage attack, not a collision attack. For a generic hash function with ℓ-bit output, an adversary making q evaluations finds a second preimage with probability at most $q/2^\ell$. Setting this $\leq 2^{-\kappa}$ gives $\ell \geq \kappa + \log_2 q$. A probabilistic polynomial-time adversary makes at most $q = \text{poly}(\kappa)$ queries, giving $\ell \geq \kappa + O(\log \kappa)$. A quantum polynomial-time adversary running Grover's algorithm finds a second preimage in $O(2^{\ell/2})$ quantum queries. To guarantee that the best quantum attack requires at least 2^κ operations, we set $2^{\ell/2} \geq 2^\kappa$, giving $\ell \geq 2\kappa$. For concrete parameters $\kappa = 256$, we obtain $\ell \geq 512$, which our 512-bit SHA3-512 output satisfies exactly. We emphasise that this bound applies to computationally bounded adversaries only. An information-theoretic (unbounded) adversary can find a preimage in f_{com} with probability 1 by exhaustive search, so no anchoring scheme can provide information-theoretic INT guarantees. Cryptographic hash functions such as SHA3-512 provide computational (bounded) security, not information-theoretic security. □

Our construction commits the 512-bit SHA3-512 output, so for $\kappa = 256$ the on-chain footprint $\ell = 512 = 2\kappa$ satisfies the quantum second-preimage bound exactly. Targeting $\kappa = 256$ requires $\ell \geq 512$ bits for second-preimage resistance via Grover's bound; SHA3-512 satisfies this exactly. Classically this provides a 512-bit security margin, future-proofing the commitment against multi-target attacks. We note that the Merkle tree exposes $(2n - 1)$ targets (each leaf and internal node hash) for a multi-target second-preimage attack, which adds a $\log_2(2n - 1)$ term to the effective query count; for $n \leq 2^{20}$ this adds at most 21 bits. While this is well within the classical security margin (which has 256 bits of slack), it slightly reduces quantum security from 256 bits to approximately 245 bits (since the quantum bound $\ell = 2\kappa$ is satisfied exactly with zero slack). Although 245 bits remains practically quantum-safe, it technically violates the strict $\ell \geq 2\kappa + \log_2 q$ requirement for $\kappa = 256$.

9.4 Selective disclosure via BBS+

The Ed25519 credential of Step 6 attests the vector ($\texttt{tx\_hash}$, R, h_1, …, h_n) atomically: verifying it requires revealing every h_i. For workloads where a publisher anchors n confidential series in one batch but wishes a consumer to verify only a subset, the

`Ed25519Signature2020` proof block may be replaced by a BBS+ multi-message signature (Boneh et al, 2004; Looker et al, 2023).

Definition 4 (selective-disclosure credential). Let $m = (m_1, \ldots, m_{n+2})$ with $m_1 =$ `tx_hash`, $m_2 = R$ and $m_{2+i} = h_i$. The publisher computes $\sigma \leftarrow$ BBS+-Sign(sk, m). The verifier generates a nonce $c \in \{0,1\}^\kappa$ and sends it to the prover. Given any disclosure set $D \subseteq \{1, \ldots, n+2\}$ chosen by the consumer, the prover produces $\pi_D \leftarrow$ BBS+-ProofGen(pk, σ, m, D, c). The verifier accepts if and only if (i) BBS+-ProofVerify(pk, π_D, $\{m_j\}_{j\in D}$, c) = 1, and (ii) the disclosed leaf hashes $\{h_i\}_{i\in D\cap\{3,\ldots,n+2\}}$, together with the necessary Merkle sibling hashes provided alongside π_D, recompute to the root $m_2 = R$. Condition (ii) ensures that the BBS+ proof attests only to the tuple as signed; the Merkle binding to the on-chain root is independently verified against the collision resistance of the hash function.

Proposition 5 (properties of the selective-disclosure credential). Under the q-strong Diffie–Hellman (q-SDH) assumption for signature unforgeability and the discrete logarithm problem (DLP) assumption for proof soundness in pairing-friendly groups, the construction of Definition 4 satisfies (i) completeness – an honest π_D always verifies; (ii) soundness – no efficient adversary produces π_D accepted for $\{m_j\}_{j\in D}$ without knowing a BBS+ signature over a vector consistent with those values; and (iii) zero knowledge, in the random oracle model – π_D leaks no information about $\{m_j\}_{j\notin D}$ beyond what is implied by their hashes already being attested.

A publisher anchoring confidential per-counterparty exposures may then disclose to a regulator only those leaves corresponding to regulated counterparties, while proving in zero knowledge that those leaves belong to the same anchored batch as the unrevealed ones; the root R and transaction hash remain disclosed and verifiable on chain. The construction is compatible with the existing `/validateSDMX` endpoint because the on-chain anchor (and therefore INT, ORG, TS and PV) is unchanged; only the in-file proof block changes.

Caveat: BBS+ INT trust assumption. Unlike the atomic Ed25519 scheme, BBS+ selective disclosure shifts the INT trust assumption: it requires trusting the credential issuer to honestly bind the leaves to R at issuance time, which fails if the credential signing key is compromised post-anchor. An adversary who steals the BBS+ key can sign a new tuple (`tx_hash`, R, h_{fake}) and generate a valid BBS+ proof accepted against the honest root R – unless condition (ii) of Definition 4 independently verifies the Merkle binding. When condition (ii) is enforced, the adversary must also find a Merkle proof that h_{fake} (or a set of forged leaves) opens to R, which requires a second-preimage break of SHA3-512; the BBS+ key alone is insufficient to break INT. By stealing the BBS+ key, the adversary projects the legitimate root R onto a forged equivalence class of leaves, severing the canonical binding between the dataset and the on-chain commitment. This illustrates that the BBS+ construction centralises the INT invariant to the credential issuer, whereas the zero-knowledge Merkle construction preserves the decentralised, immutable invariant of the on-chain root.

When condition (ii) of Definition 4 is enforced, the BBS+ signature and the Merkle recomputation are complementary layers: the Merkle recomputation provides the INT binding to R independently of the BBS+ signing key, while the BBS+ proof provides zero-knowledge selective disclosure of individual leaves. A stand-alone zero-knowledge Merkle inclusion proof offers the same INT guarantee without the pairing overhead, but cannot hide the number or position of undisclosed leaves in the way

that BBS+ can. Designers should therefore choose either (i) BBS+ with Merkle recomputation (condition (ii)), which preserves INT under key compromise while adding pairing-group overhead; (ii) standard Ed25519 combined with zero-knowledge Merkle inclusion proofs, which also preserves INT under key compromise with smaller cryptographic primitives but reveals the leaf position; or (iii) restricting BBS+ use to settings where the credential signing key is protected at least as strongly as the anchoring key, accepting the stronger trust assumption.

Remark (cryptographic maps for selective disclosure). The choice between BBS+ signatures and zero-knowledge Merkle inclusion proofs is not a gauge choice within a single equivalence class, but a selection between distinct cryptographic maps projecting the data onto structurally different trust equivalence classes. BBS+ is a map F_{BBS}: Leaves → BBS-Sig that sends the leaf vector into a multi-message signature space whose INT invariant is centralised under the credential issuer; a compromise of that issuer severs the canonical binding, projecting arbitrary forged leaves onto the honest root's equivalence class (see the caveat above). Zero-knowledge Merkle is a distinct map F_{ZK}: Leaves → Merkle-Proof that sends the leaves into an algebraic commitment space defined strictly by the collision resistance of SHA3-512, preserving the decentralised INT invariant regardless of credential key compromise. Because F_{BBS} and F_{ZK} target fundamentally different algebraic structures with non-interchangeable trust assumptions, they target different trust equivalence classes when considered without condition (ii). When condition (ii) of Definition 4 is enforced, the Merkle recomputation anchors both maps to the same INT equivalence class (SHA3-512 collision resistance), and the distinction becomes one of disclosure granularity and proving overhead rather than trust semantics.

Remark (the ledger as a canonical gauge). The same gauge philosophy extends to the blockchain layer. In a distributed system, the equivalence class of valid system states is highly redundant and subject to asynchronous local perturbations. The blockchain consensus mechanism acts as a canonical gauge projection, mapping this high-dimensional equivalence class of distributed states onto a single, totally ordered canonical timeline (the ledger sequence). Anchoring the Merkle root R to the ledger fixes the dataset's history to this canonical invariant, stripping away the local network gauge freedoms – fork, reorganisation and latency – and establishing a universally verifiable ground truth.

9.5 Multi-publisher accumulator

The single-publisher construction generalises to P publishers sharing one anchor per epoch. Each publisher $p \in \{1, \ldots, P\}$ assembles a per-publisher root $R^{(p)}$ over its own n_p series using the construction of Section 4.1. The P roots are aggregated into a super-root

$$R^* = Root(R^{(1)}, R^{(2)}, \ldots, R^{(P)}),$$

using the same tagged construction; only R^* is anchored in one XRPL transaction. An inclusion proof for series i of publisher p is the concatenation $\pi^*_{p,i} = \pi^{(p)}_i \,\|\, \pi^{(forest)}_p$, of size $\lceil \log_2 n_p \rceil + \lceil \log_2 P \rceil$ siblings. The forest-level sibling roots $\{R^{(q)}\}_{q \neq p}$ needed to verify $\pi^{(forest)}_p$ are published by the coordinator in a public registry indexed by epoch and super-root R^*; each publisher's credential attests to its own $R^{(p)}$, and the coordinator's registry is itself integrity-protected by anchoring a hash of the registry entry alongside

R^* (or by including the sibling roots in the anchoring transaction's memo when P is small enough). Verifiers resolve R^* from the on-chain memo, retrieve the per-publisher roots from the registry, and verify $\pi^{(forest)}_p$ against the published siblings.

Proposition 6 (per-publisher accountability). If each publisher p signs $R^{(p)}$ in its own credential and the super-root R^* is anchored on chain, then by Theorem 1 and Corollary 2 no publisher can be falsely implicated in series of another, and no publisher can repudiate any series whose inclusion proof verifies.

Fee economics. Let Φ be the fixed per-anchor on-chain fee. With P publishers sharing one anchor and contributing $\{n_p\}$ leaves respectively, the natural use-proportional cost allocation charges publisher p

$$\varphi_p = \Phi \cdot \frac{n_p}{\sum_{q=1}^{P} n_q},$$

which tracks each publisher's contribution to the super-root's leaf count and recovers the single-publisher fee when P = 1. The standard Shapley value of the abstract fixed-cost game $v(S) = \Phi \cdot 1[S \neq \emptyset]$ is the egalitarian split Φ/P, which ignores heterogeneous usage. We instead employ proportional cost allocation (Moulin, 2002), charging each publisher in proportion to its leaf count. This satisfies (i) budget balance $\Sigma_p \varphi_p = \Phi$; (ii) recovery of the single-publisher case at P = 1; (iii) strict monotonicity in n_p; and (iv) anonymity with respect to publisher identity given equal n_p. Unlike the egalitarian Shapley value, the proportional rule maps the cooperative game's cost allocation onto the equivalence class defined by system usage. Per-leaf on-chain cost is therefore $\Phi/\Sigma_q n_q$, strictly decreasing in P.

Remark (canonical projection of multi-publisher state). The super-root R^* acts as a canonical projection of the P-dimensional publisher state space onto a single scalar invariant. The independent credential signatures on each per-publisher root $R^{(p)}$ preserve the partition of accountability, ensuring that the projection does not collapse the distinct trust equivalence classes of the individual publishers.

9.6 Strategic batching across publishers

The cost model of Section 6 treated batching as a single publisher's choice. With P publishers sharing infrastructure as in Section 9.5, the anchoring cadence becomes a non-cooperative game.

Game. Publisher j has Poisson dataset arrival rate λ_j and waiting-cost weight $w_j > 0$, and chooses an anchoring window $T_j \geq 0$. A shared anchor is posted whenever the soonest of the P windows closes, ie every $\tau(T) = \min_q T_q$ seconds. The on-chain fee Φ is split by the use-proportional rule of Section 9.5, which in expectation charges publisher j a fraction $\lambda_j/\Sigma_q \lambda_q$ of Φ per anchor. Publisher j's expected cost per second is

$$C_j(\tau) = \frac{\Phi \lambda_j}{\tau \sum_q \lambda_q} + \frac{w_j \lambda_j \tau}{2}, \text{ie } C_j(\tau) = \text{fee rate} + \text{waiting cost rate}$$

Proposition 7 (Nash equilibrium). The game admits a pure Nash equilibrium in which the binding publisher is $j^* = \text{argmax}_j w_j$, with realised window

$T^{NE} = \sqrt{(2\Phi / (w_{(1)} \Sigma_q \lambda_q))}$, where $w_{(1)} = \max_j w_j$.

Publishers with $w_j < w_{(1)}$ play any $T_j \geq T^{NE}$ and are non-binding.

Proof sketch. Each publisher's unconstrained minimiser of $C_j(\tau)$ is $\tau^*_j = \sqrt{(2\Phi/(w_j \Sigma_q \lambda_q))}$, which is decreasing in w_j. Because $\tau = \min_q T_q$, the publisher with the largest w_j binds the cadence at $\tau^*_{j^{\wedge *}} = T^{NE}$. Any $T_j \geq T^{NE}$ by a non-binding publisher is a best response, since lowering T_j below T^{NE} strictly increases its own cost. □

Proposition 8 (social optimum). The utilitarian social optimum minimises $\Sigma_j C_j(\tau)$ and is $T^* = \sqrt{(2\Phi / \Sigma_q w_q \lambda_q)}$. Whenever Var(w) > 0, $T^* > T^{NE}$, ie Nash play over-anchors relative to the social optimum and inflates the total fee bill. The price of anarchy is bounded above by $\sqrt{(w_{(1)} \Sigma_q \lambda_q / \Sigma_q w_q \lambda_q)}$.

Numerical illustration. Let $A = w_{(1)} \Sigma_q \lambda_q$ and $B = \Sigma_q w_q \lambda_q$. A direct calculation gives the realised price of anarchy in closed form, $PoA = (A + B)/(2\sqrt{(AB)}) \leq \sqrt{(A/B)}$, where the inequality holds because $A \geq B$ by definition of $w_{(1)}$, with the bound tight only as $B/A \to 1$. Table 4 instantiates the game with P = 5 symmetric-arrival publishers (λ_j = 1 dataset/s) and the XRPL anchor fee $\Phi = f_p = 3 \times 10^{-6}$ USD of Section 6, across three weight scenarios. The bound overstates the realised price of anarchy by a factor of 2 at extreme heterogeneity, and by ≈67% at the heterogeneity levels of Table 4, while the realised T^*/T^{NE} ratio equals $\sqrt{(A/B)}$ exactly – coinciding with the price-of-anarchy upper bound (2.24 in scenario C) – enough for a coordinator to recoup non-trivial fee savings by running the VCG mechanism below.

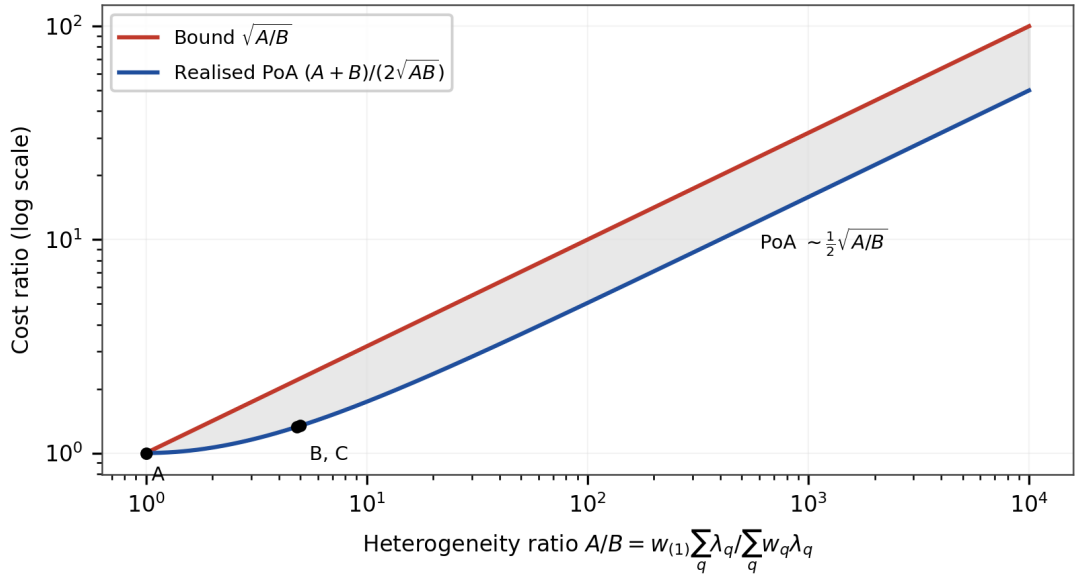
Strategic batching: Nash anchoring window, social optimum and price of anarchy Table 4

Scenario	T^{NE} (s)	T^* (s)	PoA	Bound $\sqrt{(A/B)}$
A: homogeneous, $w_j = 10^{-3}$	0.0346	0.0346	1.00	1.00
B: $w_1 = 10^{-2}$, $w_{2..5} = 10^{-4}$	0.0110	0.0240	1.32	2.19
C: $w_1 = 10^{-1}$, $w_{2..5} = 10^{-5}$	0.0035	0.0077	1.34	2.24

P = 5 publishers, λj = 1 dataset/s, Φ = 3 × 10−6 USD. PoA denotes the realised price of anarchy; the final column reports the closed-form bound of Proposition 8.

Sources: Authors

Price of anarchy in the strategic-batching game Graph 5



The realised price of anarchy grows as ½√(A/B) for large heterogeneity, so the closed-form bound √(A/B) overstates the actual cost gap by a constant factor of 2 asymptotically (and by ≈65% at the heterogeneities of Table 4). The shaded gap between the two curves is the slack that the VCG mechanism recovers.

Source: Authors

Mechanism design. A sealed-bid Vickrey–Clarke–Groves (VCG) mechanism aligns equilibrium with the social optimum. Each publisher reports $\hat{w}_j$; the coordinator computes $T^*(\hat{w})$ from Proposition 8 and charges publisher j a transfer equal to the externality that j's reported weight imposes on the others,

$$t_j = \sum_{i \neq j} C_i\left(T^*(\hat{w})\right) - \sum_{i \neq j} C_i\left(T^*(\hat{w} - j)\right).$$

Truthful reporting is dominant by the standard VCG argument, and the realised cadence coincides with T^*. The mechanism is implementable on chain: bids, allocations and transfers can themselves be credential-attested commitments anchored alongside R^*, so the coordinator's behaviour is publicly verifiable under the same trust model as the data anchors.

Remark (canonical projection). The VCG mechanism acts as a policy map, mapping the non-cooperative Nash equilibrium – a locally invariant but globally suboptimal fixed point – onto the utilitarian social optimum, thereby aligning individual strategic incentives with the collective equivalence class of Pareto-efficient outcomes.

Section summary. This section has set out a computational threat model (with Dolev–Yao network capabilities) and an explicit single-compromise resilience table; a binding theorem reducing Merkle integrity to SHA3-512 second-preimage resistance; a computational lower bound that the 512-bit anchor meets up to a small constant;

a BBS+ extension delivering zero-knowledge selective disclosure of individual series within a confidential batch; a multi-publisher accumulator with O(log n_p + log P) inclusion proofs and per-publisher accountability; and a mechanism-design result showing that a VCG-style auction over urgency weights closes the equilibrium gap between strategic batching and the social optimum.

10. Security, privacy and governance considerations

The adoption of blockchain technology for SDMx data dissemination requires balancing the benefits of transparency and immutability with the requirements for data confidentiality and institutional governance. This section examines how our system addresses these seemingly competing priorities through careful architectural design, cryptographic best practice and clear operational frameworks.

Data confidentiality. No dataset content is placed on-chain. Only cryptographic commitments (hashes / Merkle roots) are stored on XRPL. Verification reveals nothing about dataset contents beyond the ability to check the consistency of provided datasets.

Key management and trust anchors. Publisher signing keys should be managed securely. Trust anchors may be X.509 certificates or DIDs distributed via SDMx metadata endpoints or a public key registry. All key-rotation events should be recorded in metadata and optionally anchored.

Revocation and updates. If a dataset is retracted, the publisher can publish a revocation record off-chain and optionally anchor the revocation on-chain. Because the ledger is append-only, retractions are represented by follow-up commitments rather than deletion.

Formal properties. We identify the following properties provided by the system under standard cryptographic assumptions.

- **Integrity.** Assuming SHA3-512 is collision resistant (the formal assumption of Theorem 1), an adversary cannot produce a different canonicalised payload $\tilde{X}' \neq \tilde{X}$ whose Merkle root matches that of an honestly generated dataset, except with advantage negligible in the security parameter. In the practical attack model, the adversary does not control the honest dataset's hash (which is fixed before any adversarial action), so the relevant security property is second-preimage resistance of SHA3-512. Against a quantum adversary, Grover's algorithm finds a second preimage in $O(2^{\ell/2})$ queries, giving 256 bits of quantum security for ℓ = 512. Collision resistance is weaker (170 bits via the BHT algorithm at $2^{\ell/3}$), but collisions are not a valid attack vector against the INT property in our threat model, because the adversary does not control both inputs. Anchoring R on a public ledger ties the root to an immutable timestamped ledger position.
- **Non-repudiation.** Publisher-signed anchor metadata and transaction receipts provide evidence that a given publisher attested to a dataset at a specific ledger time.
- **Transparency.** Publicly accessible anchors enable third-party verification independent of the publisher.

Attacks and mitigations.

- **Second-preimage attack.** SHA3-512 mitigates second-preimage risk to cryptographically negligible levels at present (National Institute of Standards and Technology 2015; Bertoni et al, 2011).
- **Fake anchors.** Verifiers must check anchor transactions on XRPL (ledger index and transaction hash) and validate publisher signatures where applicable.[10]
- **Compromised keys.** For blockchain transaction signing keys, robust multi-signature governance with hardware security modules and secure key management practices are essential. However, we must distinguish between two key hierarchies. *Blockchain transaction keys* sign XRPL transactions and should be secured through institutional governance with multi-signature schemes, hardware security modules and secure key management practices; these keys typically have substantial financial backing and should follow strict institutional security protocols. *Publisher identity keys* sign SDMx datasets or verifiable credentials and are separate from blockchain keys; for these, key rotation is important, and they follow standard PKI practices with periodic rotation, certificate revocation lists, or distributed ledger-based key management (eg DIDs with key rotation capabilities). A lightweight on-chain attestation registry bridges the two key hierarchies: each XRPL anchoring address registers its authorised credential signing public keys via a dedicated attestation transaction, signed by the anchoring key. The registry is queryable by any verifier, allowing Algorithm 2 to confirm that the credential signing key is attested for the XRPL address that signed the anchoring transaction, without requiring the two keys to be identical. Key rotation on the identity side requires updating the registry entry; revocation is immediate once the registry update is confirmed on ledger. The system design explicitly separates these concerns: blockchain anchoring uses institutional governance for transaction signing, while publisher identity uses standard key management practices appropriate for its threat model. This separation means that rotating publisher identity keys does not require changes to smart contracts or blockchain deployment parameters; it is handled at the application layer through verifiable credentials or PKI updates. This architecture ensures operational flexibility while maintaining strong security guarantees for both layers.
- **Sybil attacks.** XRPL's consensus mechanism provides resistance; publishers should follow best practices for XRPL validators where applicable (Chase and MacBrough, 2018).
- **Data availability.** Off-chain data must remain available through redundant storage systems and/or decentralised storage.

10.1 Limitations and long-term operational risks

The architecture above is intentionally conservative, but several risks deserve explicit treatment so that adopting organisations can plan for them.

[10] XRPL.org: "XRP Ledger documentation and developer resources", https://xrpl.org/docs.

- **Erroneous or retracted publications.** An anchored fingerprint is immutable, but the dataset it represents may later be found to contain errors. The system handles this by issuing a follow-up anchor: the publisher releases a corrected dataset, anchors its fingerprint as a new entry, and publishes an off-chain revocation record (also anchored) that points from the erroneous transaction hash to the correction. Consumers verifying the old dataset still get a cryptographically valid result, but the revocation lookup tells them that the dataset has been superseded. We recommend that publishers expose a public "status" endpoint keyed by transaction hash so that revocations are discoverable without scanning the chain.
- **Dataset updates and versioning.** Routine updates (eg scheduled monthly releases of the same series) are treated as new anchors; the SDMx dataflow / dataset identifier ties successive anchors together. The system does not silently overwrite the prior commitment, so the full version history is auditable on-chain.
- **Key management responsibilities.** Two key hierarchies must be operated: institutional XRPL anchoring keys (high-value, low-rotation, ideally HSM-backed and multi-signature) and publisher identity keys for verifiable credentials (rotated on a standard PKI/DID cadence). Loss of an anchoring key blocks publication but does not invalidate prior anchors; compromise requires immediate rotation, publication of a key-rotation event in the SDMx metadata channel, and optionally an on-chain anchor of the rotation.
- **Responsibility for errors.** The technology does not change institutional accountability: the publisher remains legally responsible for the dataset's correctness. The blockchain only certifies what was published, by whom and when. Governance documentation should make this division of responsibilities explicit, especially when consumers may treat anchored data as authoritative for downstream contractual or regulatory purposes.
- **Long-term storage.** On-chain memos are durable for the life of the ledger, but the off-chain dataset payloads must be preserved separately. We recommend a tiered policy: hot storage (object store) for recent releases, cold archival storage (write-once, organisation-controlled or trusted third party) for older releases, and content-addressed mirroring (eg IPFS) for public datasets. Storage policies should be aligned with the organisation's records-retention schedule, and the hash function should be cryptographically agile (see the post-quantum item below).
- **Blockchain availability and vendor lock-in.** If XRPL becomes unavailable, prohibitively expensive or politically unsuitable, the anchoring component can be re-pointed at another ledger; only the publication pipeline going forward is affected. Past anchors remain valid as long as the historical XRPL ledger remains queryable through any archival node, and the system architecture explicitly supports multi-chain mirroring (Section 4) for organisations that wish to anchor on more than one chain simultaneously to avoid single-ledger risk.
- **Fork scenarios.** A persistent ledger fork would split history into two branches. Because anchors are short, idempotent and content-addressed, the publisher republishes the affected roots on the canonical branch chosen by the operator. Verifiers should pin to the canonical branch via published validator UNL

configuration; the verification client surfaces a clear warning if the targeted node is on a minority branch.

- **Post-quantum transition.** SHA3-512 retains post-quantum security: Grover's algorithm reduces its effective pre-image resistance from 512 to 256 bits, and the BHT algorithm reduces collision resistance to approximately 170 bits, both of which are considered quantum-safe at current security parameters (Brassard et al, 1997). However, the XRPL signature scheme (Ed25519) is vulnerable to Shor's algorithm and is not post-quantum secure. We treat the signature scheme as pluggable: when standardised post-quantum primitives (eg a lattice-based signature such as CRYSTALS-Dilithium (Bos, 2018)) become available on the target ledger, the signing pipeline can be upgraded. A caveat applies to historical data: while the on-chain Merkle roots (SHA3-512) remain secure even against a quantum adversary, the identity binding in credentials signed with Ed25519 would be retroactively forgeable by a sufficiently powerful quantum computer, so prior anchors would lose their non-repudiation guarantee even though the data integrity (the root commitment) remains intact.
- **Cost control for public deployments.** Because the reference implementation (Rusev, 2026) is openly available, any production deployment that points the anchoring gateway at a fee-bearing mainnet must bound its on-chain spend. The reported measurements use the fee-free XRPL DevNet, so the public artefact itself incurs no monetary cost; for mainnet operation we recommend an explicit per-wallet spend cap, request rate-limiting on the `/publishSDMX` endpoint, and per-publisher anchoring quotas, so that a misconfigured client or compromised credential cannot drain the institutional anchoring wallet. These controls are operational rather than cryptographic and are independent of the verification guarantees.
- **Operational and reputational risks.** A misconfigured publisher could anchor an incorrect dataset; the on-chain record then has to be explicitly retracted via the revocation flow above. Organisations should treat the anchoring pipeline as part of the dissemination critical path and include it in incident-response runbooks.

Section summary. The system relies only on commitments on-chain; sensitive content stays off-chain. Errors are handled by follow-up commitments rather than deletions, key management is separated into two distinct hierarchies, and the long-term risks (ledger unavailability, forks, post-quantum) are mitigated by the system's chain-agnostic and crypto-agile design.

11. Applicability beyond SDMx

Although the design choices in this paper are tuned for SDMX-ML, the underlying pattern – canonicalise, hash at a semantically meaningful granularity, aggregate, anchor, bind via a verifiable credential – is data-format-agnostic. We outline four concrete extensions that we believe are immediately useful and require only narrow, well defined adapters.

- **XBRL and regulatory filings.** XBRL instance documents (corporate filings, supervisory reports such as FINREP/COREP) have an existing canonical form

(OIM, c14n) and a natural per-fact granularity. Replacing the SDMx <Series>-level fingerprinting with per-fact or per-context fingerprints would let regulators and investors verify individual facts without revealing unrelated facts. The Merkle aggregation, anchoring and verification layers are unchanged.

- **CSV and tabular open data.** For tabular formats lacking a native canonical form, the adapter is a deterministic CSV serialiser (fixed column order, UTF-8, RFC 4180 quoting, normalised line endings). Per-row hashing then yields a natural granularity for selective verification of subsets of rows, which is useful for large open-data releases (eg census tables) where consumers typically slice the dataset before use.
- **JSON-based statistical APIs (SDMX-JSON, JSON-stat, REST data feeds).** Adapters convert each response to JSON Canonicalization Scheme (JCS, RFC 8785) bytes before hashing. The advantage over SDMX-ML is smaller payloads; the disadvantage is that JSON's loose typing makes the canonical form more reliant on a strict schema. Per-observation fingerprints map directly onto the existing leaf-level Merkle layer.
- **Scientific datasets and reproducibility artefacts.** For research workflows (eg environmental sensor archives, clinical trial data, machine learning training sets), the same machinery can certify both the dataset and the analysis script that produced it. Two anchors – one for the input data, one for the canonicalised script – give independent reviewers a cryptographic chain of custody for the result, complementing approaches such as data DOIs.

In each case, the only piece that has to be re-implemented is the canonicalisation rule for the target format; the Merkle aggregator, the XRPL gateway, the verifiable-credential issuer and the verification client are reusable as is. This positions the system as a generic "verifiable data product" substrate rather than an SDMx-only solution.

Section summary. The four-step pipeline (canonicalise, hash, aggregate, anchor) is format-agnostic. Concrete extensions to XBRL, CSV, JSON statistical APIs and scientific datasets require only a new canonicaliser; all other components are reusable.

12. AI agents and automated verification workflows

AI agents can be used to gather, process and disseminate or share statistical data. The system described here fits naturally into agent architectures in two ways.

1. **Verification-as-a-service for agents.** Agents can programmatically call the `/validateSDMX` API, fetch the anchored root and inclusion proof, canonicalise candidate payloads, and make a local cryptographic decision (valid/invalid). This allows agents to refuse decisions based on unverifiable data, improving downstream robustness.
2. **Autonomous monitoring and remediation.** Agents can continuously monitor SDMx endpoints, compare newly published releases with anchored metadata, run econometric anomaly detectors (eg the simple CUSUM test presented above), and open incident tickets or automatically request re-publication if mismatches are

detected. For high-assurance automation, agents must themselves be authenticated and authorised; we recommend mutual TLS or signed agent identities (DIDs) and careful key management for agent keys.

Risks and mitigations for agents. Agents introduce operational risks: leaked agent keys could be used to spam anchoring requests, and buggy agents might misinterpret canonicalisation rules. Mitigations include rate-limiting, per-agent credentials, audited agent code and signed attestations for agent actions.

Operational considerations for agent-based systems. The introduction of automated agents into official statistics workflows would require addressing several operational considerations. In environments where statistical integrity and institutional accountability are paramount, any automation must be implemented with appropriate oversight and control mechanisms. Potential considerations include:

- **Institutional governance.** Agent-based automation would need to operate within established institutional governance frameworks, with clear accountability structures and human oversight requirements.
- **Security and access control.** Implementation would require robust security measures, including rate-limiting, per-agent credential management and audit trails of agent actions, to prevent misuse and ensure accountability.
- **Operational reliability.** Automated systems would need to demonstrate operational reliability and predictable behaviour, with fallback mechanisms and manual override capabilities for critical functions.
- **Interpretation accuracy.** Any automated interpretation of canonicalisation rules or verification procedures would require extensive testing and validation to ensure consistency with manual verification processes.

These considerations highlight that, while the technical infrastructure for agent integration exists, operational deployment in official statistics would require careful planning, testing and adherence to institutional policies regarding automated systems.

Remark (algorithmic canonicalisation of agent oversight). By requiring AI agents to verify data via cryptographic inclusion proofs before execution, the non-deterministic human oversight process is mapped onto a deterministic, cryptographically verifiable equivalence class of agent states. The anchored transaction serves as a canonical fixed point, ensuring that multi-agent evolutionary topologies cannot silently diverge based on hallucinated or tampered statistical inputs.

Section summary. The verification API and inclusion proofs make agent-driven verification a natural extension; the open questions are operational (agent identity, rate-limiting, human oversight, accountability) rather than cryptographic.

13. Conclusion and future directions

We have presented a practical, standards-compliant implementation for blockchain-enhanced SDMx dissemination using the XRP Ledger (as an example). Our system addresses critical business needs by providing cryptographic proof of data integrity

while maintaining full SDMx compatibility. The implementation demonstrates how organisations can enhance trust in their statistical data without disrupting established workflows.

The business value proposition is particularly compelling for international organisations and public sector institutions that disseminate official statistics. These organisations can provide cryptographically verifiable audit trails for their data, enabling transparent traceability back to the original source. This capability addresses critical needs in cross-border data sharing initiatives, regulatory compliance reporting and inter-agency collaboration frameworks. By enabling real-time verification by partner institutions and automated systems, our solution facilitates more efficient and trustworthy exchange of statistical information between national statistical offices, central banks and international organisations. Furthermore, the system creates new opportunities for establishing standardised verification protocols in multilateral data sharing agreements and joint statistical publications. Compared with alternatives such as Content Credentials, our approach offers the advantages of decentralised verification (avoiding single points of failure), significantly lower operational costs, and seamless integration with existing SDMx infrastructure that is already widely adopted across the public sector.

Key technical achievements include:

- a prototype architecture demonstrating median publication latency in the 3–5 second range under the controlled conditions of Section 7, dominated by XRPL ledger close;
- O(|X| + log n) verification scaling (linear in payload size, logarithmic cryptographic overhead), with a constant on-chain footprint (one transaction) per anchored batch regardless of batch size;
- cost-effective batching optimised for statistical data publication patterns;
- a comprehensive security model with proper key management separation;
- an accurate cost model that properly accounts for batch-size dependencies in storage costs;
- multi-class batching optimisation for different data priority levels; and
- formal algorithm analysis with complexity bounds.

Recent research has further validated the importance of blockchain technology for enhancing data integrity in official statistics, reinforcing the practical relevance of our approach. The system provides a foundation for several important future directions:

- **Enterprise integration:** production deployment with organisations for cross-agency data sharing with cryptographic integrity guarantees.
- **Smart contract evolution:** migration to EVM-compatible sidechains for enhanced features such as access control and automated compliance, while maintaining the cost benefits of our current lightweight approach.
- **Chainlink oracle integration:** real-time data verification for decentralised and on-chain finance and other applications requiring trusted statistical data feeds.

- **Zero-knowledge proofs:** privacy-preserving validation of statistical properties without revealing underlying data.
- **Cross-chain anchoring:** multi-ledger anchoring for enhanced redundancy and trust distribution across different blockchain ecosystems.
- **AI agent ecosystems:** development of standardised verification protocols for autonomous AI systems consuming official statistics.
- **Regulatory compliance frameworks:** extension to support automated regulatory reporting with immutable audit trails.

In conclusion, this work shows that blockchain technology – when used narrowly, pragmatically and in a standards-aware manner – can address real-world data integrity challenges faced by statistical organisations today. By prioritising interoperability, operational realism and clear business value, the proposed system bridges the gap between cryptographic innovation and the practical requirements of official statistics, offering a credible pathway to more trustworthy and verifiable data dissemination.

References

Bai, J and P Perron (1998): "Estimating and testing linear models with multiple structural changes", *Econometrica*, vol 66, no 1, pp 47–78.

Bańbura, M, D Giannone and L Reichlin (2010): "Nowcasting", *ECB Working Papers*, no 1275, December.

Basseville, M and I V Nikiforov (1993): "*Detection of Abrupt Changes: Theory and Application*", Englewood Cliffs, NJ, Prentice Hall.

Bertoni, G, J Daemen, M Peeters and G Van Assche (2011): "Cryptographic sponge functions: design, analysis and applications", in *Selected Areas in Cryptography*.

Bloom, N, E L Groshen, D Hobbs and M R Strain (2026): "The value of reliable statistics", *NBER Working Papers*, no 35135.

Boneh, D, X Boyen and H Shacham (2004): "Short group signatures", in *Advances in Cryptology – CRYPTO 2004*, LNCS 3152, Springer, pp 41–55.

Bos, J et al (2018): "CRYSTALS-Dilithium: digital signatures from module lattices", *CRYSTALS project*.

Boyer, J and E Davis (eds) (2008): "Canonical XML version 1.1", *W3C Recommendation*.

Brassard, G, P Høyer and A Tapp (1997): "Quantum cryptanalysis of hash and claw-free functions", *SIGACT News*, vol 28, no 2, pp 14–19; also in *Proceedings of LATIN'98*, LNCS 1380, Springer, 1998, pp 163–169.

Brown, R L, J Durbin and J M Evans (1975): "Techniques for testing the constancy of regression relationships over time", *Journal of the Royal Statistical Society, Series B*, vol 37, no 2, pp 149–192.

Chainlink (2025): "The US Department of Commerce and Chainlink are now bringing government macroeconomic data onchain".

Chase, B and E MacBrough (2018): "Analysis of the XRP Ledger consensus protocol", *arXiv:1802.07242*.

Giannone, D, L Reichlin and D Small (2008): "Nowcasting: the real-time informational content of macroeconomic data", *Journal of Monetary Economics*, vol 55, no 4, pp 665–676.

Gross, D and C M Harris (1998): *Fundamentals of Queuing Theory*, 3rd ed, New York, Wiley.

Haber, S and W S Stornetta (1991): "How to time-stamp a digital document", *Journal of Cryptology*, vol 3, pp 99–111.

Herlihy, M (2018): "Atomic cross-chain swaps", *arXiv:1801.09515*.

Klein, L, K Fedchun and D Demirag (2022): "Investigating the use of blockchain to authenticate data from the Statistics Canada website", *Statistics Canada, Analytical Studies*.

Looker, T, V Kalos, A Whitehead and M Lodder (eds) (2023): "The BBS signature scheme", *IRTF CFRG Internet-Draft*.

Mariano, R S and Y Murasawa (2003): "A new coincident index of business cycles based on monthly and quarterly series", *Journal of Applied Econometrics*, vol 18, no 4, pp 427–443.

Merkel, D (2014): "Docker: lightweight Linux containers for consistent development and deployment", *Linux Journal*, vol 2014, no 239, art 2, May.

Merkle, R C (1988): "A digital signature based on a conventional encryption function", in *Advances in Cryptology – CRYPTO '87*, LNCS 293, Springer, pp 369–378.

Moulin, H (2002): *Axioms of Cooperative Decision Making*, Cambridge University Press, ch 6 ("Proportional allocation and the Shapley value").

Nakamoto, S (2008): "Bitcoin: a peer-to-peer electronic cash system", https://bitcoin.org/bitcoin.pdf.

National Institute of Standards and Technology (2015): "FIPS 202: SHA-3 standard: permutation-based hash and extendable-output functions".

QuickNode (2025): "Official US economic data is now onchain: explore the new public trust layer".

Recordon, D and D Reed (2006): "OpenID 2.0: a platform for user-centric identity management", *Proceedings of the Second ACM Workshop on Digital Identity Management*.

Ricciato, F, A Wirthmann, K Giannakouris, F Reis and M Skaliotis (2023): "Trusted smart statistics: motivations and principles", *Statistical Journal of the IAOS*, vol 39, no 4, pp 589–603 (reprinted from vol 35, no 4, 2019, with updated editorial matter).

Rusev, M et al (2026): "SDMX-Blockchain: a reference implementation for verifiable official statistics", open-source software repository, *Bank for International Settlements*, https://github.com/bis-med-it/sdmx-blockchain.

Sporny, M, D Longley and D Chadwick (eds) (2022): "Verifiable credentials data model 1.1", *W3C Recommendation*, 3 March.

Streamlit Docs: "Deploy Streamlit using Docker", https://docs.streamlit.io/deploy/tutorials/docker.

United Nations, Department of Economic and Social Affairs, Statistics Division (2023a): "Fundamental Principles of Official Statistics (FPOS) implementation guidelines, 2023 edition", *United Nations publication*.

United Nations, Department of Economic and Social Affairs, Statistics Division (2023b): "Fundamental Principles of Official Statistics: guidelines for implementation, 2023 edition", *Statistical Journal of the IAOS*, vol 39, no 2, pp 175–186.

Annex A: Algorithms

A.1 Publication algorithm

Algorithm 1: Publication algorithm

```
procedure PUBLISH(X)
  X̃ ← C14N(strip_anchor(X))            ▷ strip all sji:anchor elements, then
canonicalise
  h ← SHA3-512(X̃)                      ▷ fingerprint commits to statistical payload
only
  store (X, h) in object store (off-chain) and record metadata in DB
  let {S₁, ..., Sₙ} be the n time series contained in X
  if publisher selected a subset of series S' ⊆ {S₁, ..., Sₙ} then
    let m ← |S'|                        ▷ leaf count is the subset size, not the file
total n
    compute per-series fingerprints hᵢ = SHA3-512(C14N(Sᵢ)) for Sᵢ ∈ S'
    compute leaf nodes Lᵢ = H_leaf(hᵢ)
    build Merkle tree over the m leaves to obtain internal root R_int
    cache per-series inclusion proofs {πᵢ} for on-demand partial verification
    R ← H(0x02 ‖ len32(m) ‖ R_int)      ▷ wrapper binds the actual leaf count m
    tx ← XRPLGateway.createMemoTx(hex(R))
    res ← XRPLGateway.submit(tx)
    record anchor metadata in DB: (R, tx_hash, ledger_index, timestamp)
    inject anchor metadata (tx_hash, Type:Partial, R, Leaves:m, MerkleLeaves:{hᵢ})
      into a <message:Source> header with sourceID="sji:anchor"
    VC ← IssueVC(tx_hash, R, {hᵢ})       ▷ BBS+ or Ed25519 credential binding the
batch
    inject base64(VC) into a <message:Source> header with sourceID="sji:anchor"
  else                                  ▷ whole-file anchoring (n = 1)
    L₁ ← H_leaf(h)
    R_int ← L₁                          ▷ no padding needed; tree has a single leaf
    R ← H(0x02 ‖ len32(1) ‖ R_int)      ▷ domain-separated root wrapper
    tx ← XRPLGateway.createMemoTx(hex(R))
    res ← XRPLGateway.submit(tx)
    record anchor metadata in DB: (R, tx_hash, ledger_index, timestamp)
    inject anchor metadata (tx_hash, Type:WholeFile, R, Leaves:1, MerkleLeaves:{h})
      into a <message:Source> header with sourceID="sji:anchor"
    VC ← IssueVC(tx_hash, R, {h})       ▷ Ed25519 credential binding the file
    inject base64(VC) into a <message:Source> header with sourceID="sji:anchor"
  end if
  return X as publication receipt; proofs {πᵢ} are cached server-side for on-demand
    retrieval via API
end procedure
```

A.2 Verification algorithm

Algorithm 2: Verification algorithm (with credential and identity check)

```
procedure VERIFY(X)
  extract anchor_meta_information and (if present) Merkle leaves from X's
    <message:Source> headers      ▷ self-contained: all metadata embedded in X
  tx_hash ← extract from anchor_meta_information
  tx_meta ← fetch transaction tx_hash from XRPL
  R ← extract root from tx_meta.memo
  if anchor_meta_information.type = WholeFile then     ▷ whole-file gauge
    X' ← strip_anchor(X)          ▷ remove every sji:anchor <message:Source> element
    X̃' ← C14N(X')
    h ← SHA3-512(X̃')
    L ← H_leaf(h);  R_int ← L     ▷ single-leaf tree: internal root is the leaf itself
    if H(0x02 ‖ len32(1) ‖ R_int) ≠ R then return INVALID   ▷ whole-file mismatch
  else if anchor_meta_information.type = Partial then  ▷ selective-disclosure gauge
    extract series_id, leaf hash hᵢ and inclusion proof πᵢ from input
    S̃ᵢ ← C14N(Sᵢ);  h'ᵢ ← SHA3-512(S̃ᵢ)
    if h'ᵢ ≠ hᵢ then return INVALID
    R'_int ← RootInternal(hᵢ, πᵢ)   ▷ orientation bits in πᵢ handle padding
duplication
    extract m from anchor_meta_information   ▷ m is file-provided but not trusted: it
is
      ▷ bound by the domain-separated root wrapper, so tampered m yields R'' ≠ R
    if H(0x02 ‖ len32(m) ‖ R'_int) ≠ R then return INVALID
  else                                  ▷ full-leaf gauge (consumer has all series)
    parse declared ordered fingerprints {h'₁, ..., h'ₙ} and identifiers {id₁, ..., idₙ}
    extract only the declared <Series> elements {S₁, ..., Sₙ} from X by matching
      identifiers (if X contains additional undeclared series, return INVALID)
    if |{S₁, ..., Sₙ}| ≠ n then return INVALID
    for i = 1, ..., n do
      S̃ᵢ ← C14N(Sᵢ);  hᵢ ← SHA3-512(S̃ᵢ)
```

```
      if h_i ≠ h'_i then return INVALID
    end for
    recompute R' ← Root({h'_1, ..., h'_n})
    if R' ≠ R then return INVALID
  end if
  ▷ credential and identity verification (shared across all gauges)
  extract base64 VC from <message:Source> header
  VC_body ← base64decode(VC)
  if Ed25519Verify(VC_body, VC_signature, VC_publicKeyHex) fails then
    return INVALID                                  ▷ credential signature invalid
  if VC_body.txHash ≠ tx_hash then return INVALID     ▷ binds to wrong transaction
  if VC_body.merkleRoot ≠ R then return INVALID       ▷ binds to wrong Merkle root
  on_chain_address ← tx_meta.Account   ▷ transaction sender; SigningPubKey may differ
  registered_keys ← AttestationRegistry.lookup(on_chain_address)
  if VC_publicKeyHex ∉ registered_keys then
    return INVALID                    ▷ signing key not attested for this XRPL
address
  return VALID
end procedure
```