Open DSU

OpenDSU

ESSENTIAL PHILOSOPHY

A Conceptual Framework for Verifiable Digital Objects, Content Transfer, and AI Security

DSU • KeySSI • Nanoledger • Microledger

Future Steps

Author's Note

This work was created under the umbrella of Axiologic Research as part of two interconnected research programs, one dedicated to Artificial Intelligence and the other to Outfinitism, both led by Sînică Alboaie. To explore the details of how these two programs intersect and build upon each other, we invite readers to visit the Outfinity Venture Studio page at www.outfinity.ch.

While Artificial Intelligence language models were used to assist with text generation, the foundational philosophy, thematic structure, and analytical approach derive exclusively from the author's decades of dedicated study of social phenomena, social technologies, and genuine hard technologies resulting from various European and self funded research projects. Recently, within the Achilles research project, we have been deeply studying AI generated literature and its automated verification using neuro symbolic approaches; all the free books made available to the public are part of the suite of works we use to test our latest technologies.

Source and scope note

This book presents OpenDSU primarily as a **conceptual framework** for the design of controlled, reconstructable, and verifiable digital objects. OpenDSU has concrete open-source implementations, published RFCs, deployment experience, and a substantial technical vocabulary. Those implementations are important because they demonstrate feasibility and expose design trade-offs. They are not treated here as the final or exclusive form of the framework.

The stable subject of the book is the relation among bounded data, cryptographic authority, protected storage, verifiable history, provenance, validation, discovery, and governance. Libraries, programming languages, ledger technologies, storage services, identifier encodings, and execution environments may change. The framework remains useful only if its core concepts can survive such changes and can be implemented more than once.

The source corpus includes the *OpenDSU Yellow Book: Concepts and Design*, the current public OpenDSU documentation (www.opendsu.org), the LightDSU and LightDSU Provenance design notes, the research article on Self-Validating Data, and the Axiologic Research report on OpenDSU and tokenization. The latter two documents clarify two relationships that are used consistently throughout this edition. An **anchor is also a nano-ledger**: it is the compact object-level ledger identified by an AnchorID and extended by signed append operations. **Self-Validating Data is a possible implementation of a micro-ledger**: it binds a history of commands to replay and validation rules so that a receiving environment can reconstruct and evaluate state.

These definitions also establish a useful separation. A nano-ledger records the minimal continuity of a DSU or another bounded object, normally through signed version commitments and control transitions. A micro-ledger can admit a richer and domain-specific vocabulary of operations. An SVD is one way to implement that richer history by combining commands, provenance, state-transition rules, and replay semantics. A nano-ledger can commit the state of a micro-ledger without placing the complete micro-ledger history in a public or shared ledger.

The book does not assume that every DSU must contain an SVD, that every micro-ledger must use one history representation, or that every anchor must be stored in a blockchain. The OpenDSU public documentation explicitly allows heterogeneous ledgers and describes blockchains primarily as notarization

mechanisms for DSUs. LightDSU further demonstrates that the initial continuity record can be local and that stronger external witnessing can be added according to risk.

The status of individual RFCs is treated as source metadata rather than as a claim that the conceptual framework is complete. Anchoring and the central DSU concepts have accepted implementations. Self-Validating Data remains a research and review direction in the public corpus. Versionless objects, optimistic anchoring, key rotation, delegation, contextual mounting, personas, and related mechanisms show how the framework continues to evolve.

AI security is presented as an active research area, not as a solved application domain. Autonomous agents increase the importance of identity, authorization, sequence, provenance, and evidence latency because they can read external data, call tools, alter state, and communicate with other agents at machine speed. Current work on AI-agent standards, agent identity and authorization, adversarial use of AI, transparency services, and content provenance confirms that these questions are being addressed across several technical communities. This book proposes how OpenDSU concepts can contribute to that research; it does not claim that an anchor, a signature, or a provenance record is sufficient to make an AI system safe.

The text follows a general **Why, How, What** structure. Part I explains why conventional content systems are insufficient in adversarial and autonomous environments. Part II explains how the OpenDSU conceptual framework separates and recombines its core concerns. Part III describes what kinds of transfer systems, governance models, reference implementations, and research programmes may follow.

The central limitation is maintained throughout: cryptographic evidence can establish continuity, attribution, and declared process. It cannot by itself establish factual truth, institutional legitimacy, scientific validity, fairness, or legal acceptability. Those conclusions require domain evidence and contestable governance.

Contents

This book presents OpenDSU as a conceptual framework and research programme, not as an implementation manual.

PREFACE

Why OpenDSU Should Be Read as a Conceptual Framework

OpenDSU emerged through a sequence of research projects, enterprise collaborations, open-source implementations, and RFCs. Its terminology therefore contains both durable concepts and historical implementation choices. The Yellow Book had to explain a complete technology environment: Data Sharing Units, KeySSIs, Brick Storage, BrickMaps, anchoring, hierarchical ledgers, BDNS, reconstruction, mounting, wallets, enclaves, Self-Sovereign Applications, DIDs, credentials, APIHub, build systems, and deployment practices. That breadth was appropriate for demonstrating that the architecture could be built.

An essential-philosophy book has a different responsibility. It must identify which relationships remain necessary when the original libraries, services, and deployment patterns are replaced. A concept is not essential because it appeared early or because it has a familiar name. It is essential when removing it changes the security, governance, or portability properties that the architecture is intended to provide.

The framework can be summarized through a set of persistent relationships.

Persistent relationship	Reason it matters
Bounded information and identity	A digital object needs a stable accountable boundary so that content, versions, authority, provenance, and policy can be related without depending on one application database.
Confidential content and public commitments	High-volume or sensitive information should remain encrypted and outside shared ledgers, while compact commitments support integrity checks, ordering, and independent witnessing.
Authority and capability	The right to read, modify, delegate, or verify should be represented explicitly and attenuated rather than inherited from a broad platform account.

Persistent relationship	Reason it matters
Anchor and nano-ledger	The anchor is the compact per-object ledger that records signed version commitments and control transitions under a stable AnchorID.
micro-ledger and validation	Rich state transitions require a scoped programmable history; Self-Validating Data provides one possible micro-ledger pattern through commands, replay, provenance, and rules.
Storage and reconstruction	Custody of encrypted pieces should be separable from the authority to interpret them; an authorized resolver reconstructs only the view permitted by capability and policy.
Domain name and trust configuration	Identifiers must resolve across heterogeneous stores, witnesses, ledgers, organizations, and jurisdictions without pretending that every domain has one global trust model.
Application and security context	Data control is incomplete unless the code that uses keys and interprets information is constrained by an explicit execution and policy environment.

This conceptual reading changes the role assigned to blockchain. OpenDSU does not require a universal ledger containing the world state. A shared ledger is one possible witness. Its useful function is narrower: register commitments, improve freshness guarantees, expose equivocation, and make later rewriting more difficult. The confidential content, the domain rules, and much of the computational validation can remain outside the witness.

It also changes the role assigned to the DSU. A DSU is not simply an encrypted archive. It is a bounded logical object whose protected content can be reconstructed from content-addressed pieces, whose authority can be represented through capability families, and whose evolution can be related to an anchor or nano-ledger. Some DSUs may additionally contain or mount micro-ledgers, validation code, credentials, or application logic. Others may deliberately expose only the current state. The framework should make these temporal and semantic differences explicit.

The research on Self-Validating Data extends the same reasoning. Conventional data is passive: an application supplies the rules, the history, and the interpretation. An SVD binds a data structure to the commands and validation logic needed to derive a valid state. The SVD can be stored locally, exchanged peer to peer, or anchored indirectly. Its significance is not a particular glue language or blockchain adapter. Its significance is the possibility that domain rules and provenance remain attached to the history they govern.

LightDSU provides a complementary test. It asks whether the OpenDSU relationships can be implemented as a small local library without requiring a distributed ledger in the core. Its "lkey", "rkey", and "lza" capabilities preserve control, read, and verification-only authority. Its encrypted BrickMap and encrypted bricks preserve reconstruction. Its EventSSI sequence records versions, grants, revocations, access, provenance, policy, and key epochs. The local implementation does not provide independent consensus, but it produces an object whose history can later receive organizational, consortium, transparency-service, or public receipts.

This is especially relevant to AI security. Autonomous agents are not merely content generators. They combine data, models, tools, external services, memory, and delegated permissions in order to change external state. A malicious or compromised agent can exploit valid but stale information, inconsistent histories, delayed revocation, ambiguous identity, or a short interval between an action and independent evidence. The relevant defense is not an assertion that every AI output is true. It is an architecture in which consequential operations can be related to exact inputs, authority, prior state, policy, tool activity, and durable receipts.

The chapters that follow do not prescribe a final OpenDSU implementation. They define a technical research position: future digital systems will require inexpensive, composable ways to preserve continuity, attribution, and provenance while maintaining confidentiality and allowing heterogeneous governance. OpenDSU offers a coherent conceptual vocabulary for that problem.

PART I

Why Verification Must Become Native

CHAPTER 1

Autonomous Systems and the Verification Gap

Frame	Chapter focus
Why	Autonomous agents can turn plausible but stale, reordered, or insufficiently authorized content into external action before human review.
How	The chapter separates payload authenticity from continuity, authority, provenance, and witness evidence, then places verification before consequence.
What	A verification-oriented workflow produces portable signed events and a bounded verification report for each material transition.

Digital systems were designed under an assumption that is becoming progressively less reliable: content moves faster than verification, but human review can eventually restore context. A document can be emailed without its complete history because a person may telephone the sender. A database extract can lose provenance because an auditor can request logs. A decision can be recorded as a final value because the responsible team remains available to explain how it was reached.

Autonomous AI systems weaken this assumption. An agent can receive information, choose a tool, invoke an external service, update a record, communicate with another agent, and trigger a financial or operational consequence before a human understands that the workflow has started. The principal security problem is therefore no longer confined to whether a message is authentic. The system must determine whether the proposed action belongs to the accepted history, whether the authority remains current, and whether the evidence is sufficient for the consequence.

Consider a clinical decision workflow in which a hospital agent produces a medication recommendation. The recommendation may be syntactically correct

and signed by a legitimate service. It may still be unsafe because the agent used a superseded patient record, a model version withdrawn after a safety incident, or an authorization that had been revoked seconds earlier. A receiving pharmacy system that verifies only the signature will accept a correctly signed but contextually invalid act.

The verification gap can be expressed as a difference between the information available to the acting system and the information required to justify the action.

Information available at action time	Information required for justified reliance
Payload and sender identifier	Exact content commitment, canonical representation, and a verifiable relation between the sender identifier and the signing authority.
Current application state	Prior accepted object state, branch continuity, and evidence that the state has not been superseded.
Access token or session	Capability scope, purpose, authority epoch, delegation path, and revocation status.
Model or tool label	Immutable or otherwise accountable artifact identity, configuration, dependencies, and the provenance of external tool results.
Application log	Portable event evidence that remains interpretable if the original platform, operator, or vendor becomes unavailable.
Timestamp	Monotonic object sequence, independent receipt time, freshness policy, and an explicit account of disconnected or provisional operation.
Output explanation	Commitments to inputs, policy, transformations, and approvals that can be evaluated independently of fluent narrative.

The gap is not specific to machine learning. It exists in supply chains, digital identity, clinical research, financial messaging, software distribution, scientific publication, and public administration. AI increases its severity by reducing the time available for correction and by making plausible content inexpensive to produce. A malicious agent may not need to forge a durable record. It may need

only to make a valid-looking alternative arrive first and remain unchallenged long enough for another system to act.

The same property makes conventional logs insufficient. Logs are usually controlled by the service whose behavior they are intended to explain. They may be mutable, incomplete, inaccessible to another organization, or expressed in a format that is meaningful only to the original software. Even when they are retained honestly, they often record technical events rather than the semantic act: a file read rather than the declared purpose of an analysis, an API call rather than the authority under which a decision became operational.

The OpenDSU response is to move essential evidence closer to the information object and the act. A consequential change should create a signed event connected to the preceding accepted state. The event should commit to the resulting version, the controlling authority, the operational actor when distinct, the applicable policy, and any detailed provenance object required by the domain. An external witness may then register the commitment at the assurance level justified by risk.

This approach is sometimes described as notarization by default. The phrase must be interpreted precisely. It does not require publication of confidential data. It does not mean that every read operation deserves a permanent global record. It means that a material transition should normally produce evidence as part of the transaction rather than as an optional compliance process added later.

Materiality is domain dependent. A local note may need only a local signed history. A cross-institution clinical result may require a consortium receipt and a regulated provenance profile. A public safety directive may require several independent witnesses and threshold approval. The framework supplies a common form for producing and escalating evidence; policy determines the required level.

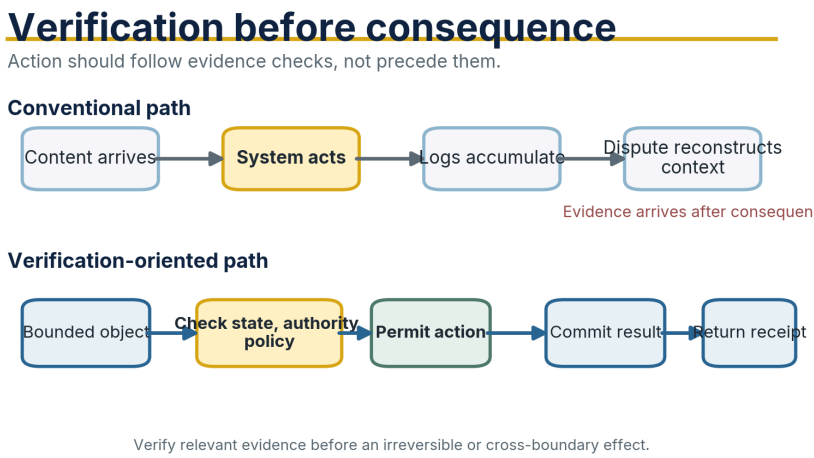

Figure 1. The verification gap arises when an autonomous decision can trigger external action before current evidence is checked.

The receiving system should evaluate the evidence before consequence. This shifts the order of operations. Conventional systems often act and then audit. A verification-oriented system first resolves the object, checks the anchor or nano-ledger, establishes the capability and authority epoch, evaluates required provenance, and determines whether the witness state satisfies policy. Only then does it reconstruct protected content or permit an external side effect.

Human oversight remains necessary, but it becomes more specific. Instead of asking a reviewer to infer trust from a narrative, the system can present a verification report that separates what was established from what remains uncertain. A valid signature, a current branch, an accepted provenance profile, and an independent receipt can be shown independently. The reviewer can then focus on domain judgment, fairness, and exceptional circumstances.

The verification gap cannot be eliminated entirely. Sensors can be wrong, actors can collude, institutions can be corrupt, and validators can contain defects. The architectural objective is narrower and measurable: prevent valid-looking content from becoming operational merely because the surrounding history, authority, and provenance are difficult to obtain in time.

The need for this objective explains why OpenDSU should be understood beyond the technology cycle in which it first appeared. Its essential problem is not enterprise blockchain adoption. It is the design of information that remains

verifiable when content, infrastructure, and autonomous actors move independently.

CHAPTER 2

Digital Sovereignty as Accountable Control

Frame	Chapter focus
Why	Possession of a copy or physical custody of a server does not establish effective control over data, identity, policy, and history.
How	Digital sovereignty is analysed as granular authority, replaceable dependencies, controlled disclosure, portability, and contestability.
What	The framework supports personal, organizational, and ecosystem sovereignty without requiring technical isolation.

The history of computing has often confused possession with control. A person is said to possess data because a copy appears on a screen. A company is said to own data because it pays for the database. A state is said to exercise sovereignty because servers are located inside its borders. None of these conditions is sufficient.

Real control asks harder questions. Who can decrypt the data? Who can change it without detection? Who can deny access? Who can silently change the access policy? Who can observe the metadata? Who can move the data to another provider? Who can prove which version existed before a dispute? Who can revoke a delegated capability without destroying the entire system?

Digital Sovereignty begins when those questions are treated as architectural questions rather than contractual promises.

The phrase is sometimes misunderstood as digital isolation: every person storing everything alone, every organization attempting complete technical isolation, every country cutting itself off from the world. That would be a fragile and impoverished interpretation. Sovereignty does not mean refusing exchange. It means entering exchange without surrendering the basic ability to understand, authorize, and contest what happens to one's information.

A sovereign person may choose a cloud provider. A sovereign company may participate in a consortium. A sovereign state may rely on international infrastructure. The decisive issue is whether the dependency is visible, replaceable, and limited. Sovereignty is compatible with delegation; it is incompatible with invisible capture.

OpenDSU was built around this distinction. Its early language emphasized that individuals, enterprises, organizations, and states should retain as much control over data as possible and should avoid unnecessary intermediaries. The important word is not *all*. Absolute control is rarely possible and often undesirable. A hospital cannot allow a patient to rewrite a diagnosis. A company cannot let an employee erase regulated records. A court cannot accept that every party has a private truth. Sovereignty must exist inside governance.

This leads to three levels of digital sovereignty. The first is **personal sovereignty**. The individual should be able to hold keys or authorize a trusted agent, disclose only what is needed, understand the purpose of use, move data when possible, and preserve evidence of consent and revocation. Personal sovereignty is weakened when identity, data, and applications are all trapped behind one platform account.

The second is **organizational sovereignty**. A company, university, hospital, laboratory, or public agency must be able to protect trade secrets and regulated information while collaborating with others. Organizational sovereignty requires cryptographic boundaries, auditable delegation, and the ability to change infrastructure without rewriting every application.

The third is **ecosystem sovereignty**. When independent parties form a digital trust ecosystem, no single participant should be able to rewrite the shared history or redefine the meaning of evidence unilaterally. The ecosystem needs common rules, but those rules should not require a central custodian to see all confidential data.

These levels can conflict. A patient's desire for control can conflict with a hospital's duty to preserve a medical record. An employee's privacy can conflict with an organization's fraud investigation. A state's lawful access requirements can conflict with a citizen's fear of abuse. OpenDSU does not remove these conflicts. It offers a way to make them more explicit by separating content, keys, capabilities, histories, and witnesses.

This is why the principle **no size fits all** matters. The same architecture can support an edge wallet controlled by a person, a cloud agent controlled by a

company, or a mixed arrangement in which sensitive keys remain local while availability services remain remote. The correct design depends on threat, law, operational capacity, and consequence.

Sovereignty is therefore not a product feature. It is a distribution of power.

A system is more sovereign when authority is granular, dependencies are replaceable, disclosure is minimized, histories are portable, and disputes can be examined without begging the original platform for permission. A system is less sovereign when one provider combines identity, storage, policy, analytics, and adjudication in a form that cannot be independently verified.

The difference can be illustrated by two superficially similar links.

The first is a conventional cloud-sharing link. It identifies a resource inside a provider's namespace. The provider decides whether the link resolves, which account is recognized, what version is served, and what metadata is retained. The user may have excellent convenience and weak sovereignty.

The second is a capability derived from a self-certifying identifier. It can carry or resolve the cryptographic material needed for a defined level of access. The storage provider can make the content unavailable, but it cannot silently transform ciphertext into another valid history. A different resolver or replica may reconstruct the same object. The user may still depend on infrastructure, but the dependency has become narrower.

That narrowing is the practical meaning of disintermediation. It does not abolish service providers. It reduces the number of things they must be trusted to do honestly.

Technical point	Explanation
Sovereignty and delegated infrastructure	A sovereign system may use clouds, consortiums, identity providers, and public infrastructure. The test is whether those services are delegated utilities or unchallengeable authorities.

In the age of autonomous AI, sovereignty becomes more urgent because agents will act through permissions that humans cannot inspect transaction by transaction. A human may authorize an agent to negotiate contracts, access a medical history, operate a factory line, or coordinate scientific data. The agent's authority must be

specific, derivable, revocable, and visible in the evidence trail. Otherwise convenience turns into ambient power.

Everything else in this book follows from that definition.

CHAPTER 3

Trust as a Set of Verifiable Properties

Frame	Chapter focus
Why	Terms such as trusted platform or trustless ledger conceal different questions about integrity, identity, provenance, freshness, authorization, and governance.
How	The chapter decomposes trust into evidence types and identifies which technical or institutional mechanism can answer each question.
What	A verifier receives a bounded report of established properties and unresolved dependencies rather than one undifferentiated trust label.

Digital systems often speak about trust as if it were a substance that can be added by branding. A platform is trusted. A certificate is trusted. A model is trusted. A ledger is trustless. The language hides more than it reveals.

Trust is not one property. It is a decision made under several kinds of uncertainty.

When a system receives a piece of information, several distinct questions must be answered.

Trust dimension	Question the verifier must answer
Integrity	Has the content or canonical object changed?
Authenticity	Which key controlled the signature, and how is that key related to the claimed actor?
Provenance	Through which entities, activities, tools, models, and transformations did the object pass?

Trust dimension	Question the verifier must answer
Freshness	Is this the current acceptable state, or a replay of a valid but older state?
Authorization	Was the actor permitted to perform this operation at that sequence, time, and authority epoch?
Semantic validity	Did the transition satisfy the rules of the relevant domain?
Availability	Can the content, history, rules, and witness evidence still be obtained when verification is required?

Availability deserves special emphasis. Evidence that existed once but can no longer be resolved has little operational value. Long-lived systems need **proof liveness**: durable access to receipts, archived validators, resolver configuration, and the cryptographic material required to interpret historical commitments.

A blockchain can help with integrity and ordering. A digital signature can help with control attribution. A provenance profile can describe transformations. A policy engine can evaluate authorization. A domain validator can determine whether a clinical, financial, or scientific operation is meaningful. Replication can improve availability. None of these alone deserves the word trust.

The essential OpenDSU move is to decompose trust into inspectable evidence.

This decomposition protects the framework from two opposite errors. The first error is naive centralization: "the server says so." A central server may be competent and honest, but when the same system stores the content, defines the identity, enforces the policy, serves the history, and decides the dispute, the user cannot distinguish service from authority. Trust becomes total and opaque.

The second error is naive decentralization: "the ledger says so." A ledger can preserve an assertion without proving the assertion true. It can record a forged laboratory result with perfectly durable recording. It can establish that a key signed a statement, but not that the key belonged to the person claimed, that the instrument was calibrated, or that the method was scientifically valid.

This distinction will recur throughout the book:

Proof of history is not proof of truth. An anchor tells us that a commitment existed by a certain point and that later alteration is detectable. A signature tells us

that a particular key authorized a statement. Provenance tells us which process is claimed to have produced the result. Governance tells us why we should accept the key, process, witness, or policy. Truth remains a judgment made from evidence, domain knowledge, and accountable institutions.

Modern standards increasingly converge on this layered view. Content provenance systems attach signed manifests to media. Transparency architectures register signed statements and return receipts. Verifiable credentials separate issuer, holder, and verifier. Provenance models distinguish entities, activities, and agents. These systems do not eliminate trust. They relocate it from invisible custody to explicit claims and verification procedures.

OpenDSU belongs to the same family of thought, but with a distinctive emphasis: the evidence should remain connected to an encrypted, reconstructable unit of data whose access can be cryptographically controlled.

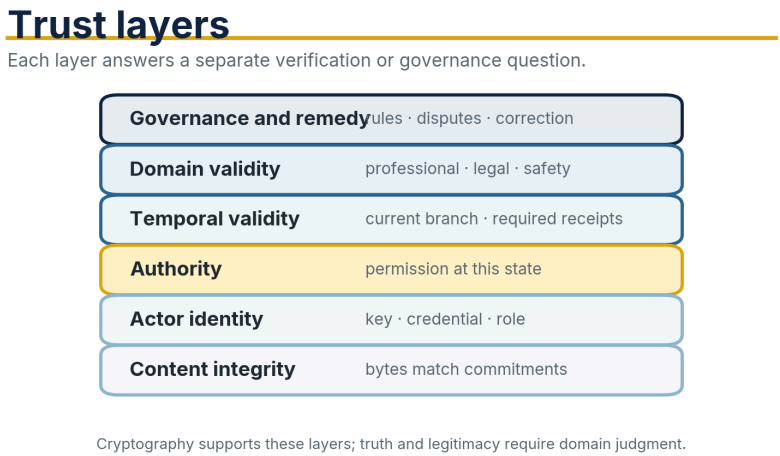

Figure 2. Trust is evaluated through distinct questions about integrity, continuity, authority, provenance, domain validity, and governance.

A useful way to think about digital trust is as a stack of questions.

At the bottom is the **content commitment**: what exact bytes or canonical semantic object are we discussing?

Above it is the **history link**: which prior state does this state claim to follow?

Above that is the **actor evidence**: which key, identity, device, organization, or software agent initiated the event?

Then comes **authority**: what permission or mandate did that actor possess?

Then comes **domain validation**: what rules make the action acceptable in this context?

At the top is **governance**: who defines the rules, how are disputes resolved, how can errors be corrected, and what remedies exist?

A system can be strong at one layer and weak at another. A perfectly signed decision made under an unjust policy is still unjust. An accurate statement with no durable history may be impossible to defend later. A correct policy executed by a compromised key may produce an invalid act. Good architecture makes these differences visible.

This visibility matters because AI systems are exceptionally good at producing the *appearance* of trust. They can generate polished explanations, plausible citations, coherent timelines, and emotionally convincing language. A human reader may be persuaded by fluency long before a machine verifier has checked the evidence. In an adversarial environment, presentation becomes part of the attack surface.

The response cannot be to distrust all content. Permanent suspicion is not governance. The response is to make certain trust questions cheap enough to answer automatically.

A receiving agent should be able to verify the content hash, signature chain, sequence, capability, anchor receipt, and provenance profile before it allows a high-impact action. A human should be able to inspect a concise explanation of what was verified and what was not. When a claim is disputed, the parties should be able to reconstruct the evidence without depending on the accused intermediary.

This is the difference between **trust by reputation** and **trust by accountable continuity**.

Reputation remains important. Institutions, experts, and long-lived identities matter. But reputation should attach to evidence-producing behavior. A hospital is trusted partly because its instruments, staff, procedures, and records can be audited. A scientific result is trusted partly because the method and lineage can be examined. A digital ecosystem should make that examinability native.

CHAPTER 4

Notarization by Default

Frame	Chapter focus
Why	Evidence assembled after a dispute is often incomplete, platform-dependent, or too late to prevent an autonomous action.
How	Material transitions create canonical signed commitments whose assurance can progress from local continuity to independent witnessing.
What	The resulting architecture distinguishes local, pending, witnessed, and policy-final states without exposing confidential content.

A digital operation becomes difficult to audit when evidence is produced only after a dispute. The application has already changed, logs have rotated, service accounts have been reassigned, and the relevant participants may disagree about which state existed. Notarization by default addresses this problem by making the creation of verifiable evidence a normal side effect of a material state transition.

The concept has three separable meanings. The first is **commitment**: the system derives a cryptographic representation of the state or event. The second is **continuity**: the new commitment is related to the preceding accepted entry in the object history. The third is **witnessing**: one or more systems outside the immediate execution path register or checkpoint the commitment according to policy.

These meanings should not be collapsed. A local signed event provides commitment and continuity without independent witnessing. A receipt from an organizational transparency service adds separation from the application. A consortium or public ledger may provide stronger resistance to unilateral rewriting. The event can remain the same while the assurance matures.

Evidence state	Meaning
Local commitment	A signed event exists in the object history and can be checked against the preceding entry. It is useful but still controlled by the local environment.
Pending witness	Local validation has succeeded and an external registration has been requested, but the receipt is not yet available. The operation must remain explicitly provisional.
Witnessed state	The required domain witness has returned evidence that it registered the commitment under its ordering and consistency rules.
Independent checkpoint	A selected event or aggregate root has been registered by an institution or infrastructure operator outside the primary domain.
Policy finality	The evidence satisfies the domain-specific rule for operational reliance. This may require one receipt, several receipts, a delay, threshold approval, or no external witness at all.

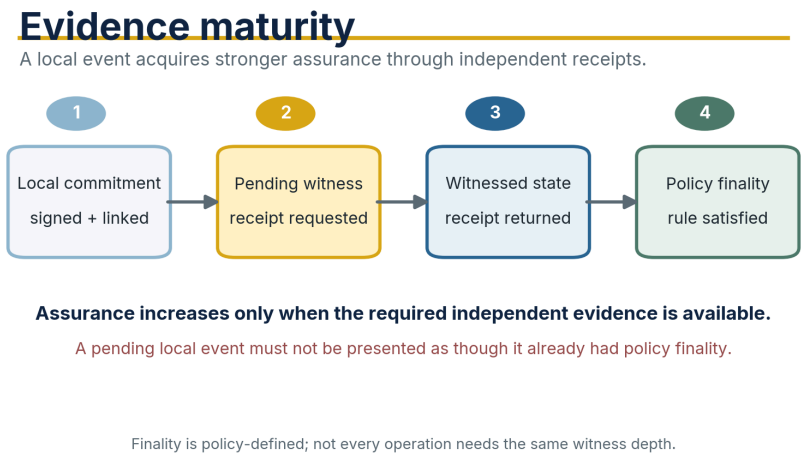

Figure 3. Evidence maturity is explicit from local commitment to independent checkpoint.

The architecture must preserve the difference between evidence creation and evidence maturity. Optimistic anchoring is useful when local systems cannot wait for an external ledger, but it is safe only if the pending state is visible and if later witness failure has a defined consequence. A provisional event should not be displayed or consumed as though it already had the assurance required for a regulated or irreversible action.

Notarization by default does not imply that every event belongs in a public blockchain. A shared ledger may store only an AnchorID and compact version commitments. Detailed provenance can remain encrypted in bricks. Routine local events can be batched. Sensitive identifiers can be represented through scoped or keyed hashes. The objective is to preserve a verifiable relation, not to maximize disclosure.

The relevant operation should also be semantic rather than mechanically exhaustive. Recording every memory read, byte copy, or internal function call would create noise, privacy leakage, and verification cost. A policy may instead require events for creation, version commit, ownership or control transition, grant, revocation, export, model execution, approval, rejection, or policy update. The event type should answer a governance question that another party may later need to verify.

A useful design separates the evidence fields from the private payload.

Compact event field	Purpose
Stable object identifier	Relates the event to the same bounded information object across storage and application changes.
Sequence and predecessor commitment	Establishes the claimed branch and makes reordering or silent insertion detectable.
State or payload commitment	Binds the event to the exact BrickMap, content version, provenance object, or result being discussed.
Control evidence	Shows which key or capability admitted the event to the object history.
Actor evidence	Identifies, directly or through a privacy-preserving reference, the person, device, service, or agent that initiated the semantic operation.

Compact event field	Purpose
Policy and authority epoch	Identifies the rules, delegation state, and key period under which the event was evaluated.
Witness receipt reference	Connects the event to external ordering, transparency, or timestamp evidence when required.

The compact record should be canonical. Independent implementations must produce the same commitment for the same event semantics, or at least identify the canonicalization method. Otherwise the architecture becomes dependent on one serializer and cannot support durable verification.

The witness need not understand the protected content or the business rule. Its responsibility is to register a commitment, preserve order, and provide a receipt. The receiving environment applies domain rules after obtaining the relevant capabilities. This separation allows confidential validation and avoids forcing every participant to expose data or logic to a global execution platform.

The limitations are fundamental. A notary can establish that a commitment was registered, but not that the underlying statement was true. A stolen control key can authorize a false transition. Colluding actors can create a consistent but deceptive history. A sensor can produce incorrect data. The value of notarization is that later substitution, denial, and selective omission become more difficult and more detectable.

The framework therefore treats notarization as one layer in a wider decision. The recipient verifies cryptographic integrity, branch continuity, authority, domain validity, provenance, and witness maturity separately. Governance defines how an invalid or disputed event is suspended, corrected, superseded, or retained as an alternative claim.

Notarization by default changes institutional practice because it moves evidence creation into the operational path. Provenance is no longer a form completed after publication. A transfer receipt is no longer available only from a platform database. A policy update is no longer an unversioned administrative action. Material operations produce the evidence needed to interpret them while the relevant context still exists.

PART II

How the OpenDSU Framework Organizes Trust

CHAPTER 5

Data Sharing Units as Bounded Verifiable Objects

Frame	Chapter focus
Why	Files, database rows, URLs, and messages are too weak as accountable exchange units because their history, authority, and policy are easily detached.
How	A DSU binds a stable logical boundary to protected structure, capability, provenance, temporal mode, and compact continuity evidence.
What	The result is a reconstructable object that can remain local or cross organizational boundaries without losing its identity and evidence relationships.

The basic unit of the web is often treated as a resource identified by a URL. The basic unit of a database is a row, document, object, or graph node. The basic unit of a filesystem is a file. These units are useful, but they are usually too thin for accountable exchange.

A file can be copied without its history. A database row can be exported without its policy. A URL can resolve to different content over time. A message can be forwarded without the context in which it was authorized. The content survives; the responsibility dissolves.

A Data Sharing Unit begins from a different question: what must travel together so that information remains meaningful, controllable, and verifiable outside its original application?

The answer is not a fixed schema. It is a conceptual envelope.

A DSU can bind several kinds of meaning without requiring them to live in one physical file.

Element	Role inside the accountable unit
Confidential content	The records, files, models, messages, or data structures whose disclosure remains controlled.
Protected structure	A versioned map or manifest that makes the unit reconstructable and gives paths and dependencies stable meaning.
Stable identity	An identifier that survives movement between stores, applications, and witness technologies.
Capability family	Distinct authority for control, reading, constrained operation, and verification without disclosure.
Evidence history	An externally checkable sequence of commitments to material states and events.
Provenance	Evidence about creation, transformation, use, approval, and derivation.
Consent and policy	The conditions under which content may be used, shared, retained, or destroyed.
Validation context	Rules, code commitments, or references needed to evaluate whether transitions are acceptable.
Mounted units	Related objects that preserve their own identity, history, and authority while appearing in a composed view.
Reconstruction metadata	The information required to locate, verify, decrypt, and assemble the authorized representation.

Not every DSU requires every element. The point is that the object has enough boundary to become accountable.

This boundary produces two complementary views. From the inside, a DSU resembles a small filesystem or private data environment. It can contain folders, records, media, configuration, credentials, and code. The user or agent reconstructs this environment only after resolving the correct capability and verifying the relevant history.

From the outside, a versioned DSU is associated with an anchor or nano-ledger. Observers may see only an identifier and a chain of cryptographic commitments. They may be able to verify that versions exist, that the sequence has not been silently rewritten, and that authorized keys approved the transitions, without reading the content itself.

The protected object and its compact history are therefore distinct but inseparable parts of the same accountable unit.

This separation of protected state from verifiable continuity is central to OpenDSU.

Placement is part of meaning. The Yellow Book distinguished on-chain, near-chain, and far-chain information. The distinction remains useful because the same bytes acquire different assurance properties depending on how closely they are bound to the evidence system.

Placement	Architectural meaning
On-chain	Compact commitments or public records are stored directly in a shared ledger. They gain common visibility and ordering, but sacrifice privacy, flexibility, and often performance.
Near-chain	Confidential or high-volume content remains outside the ledger but is deliberately and continuously bound to it through anchors. This is the natural home of reconstructable DSUs.
Far-chain	Ordinary databases, indexes, vector stores, dashboards, and analytical systems are related only indirectly, or not at all, to the anchored object. They may be useful integrations, but they must not be mistaken for equivalent evidence.

Near-chain is not a synonym for every off-chain database. It describes a deliberate evidentiary relationship: the external content can be matched to commitments, history, authority, and policy. Far-chain systems can consume or derive data, but their outputs need explicit provenance before they re-enter the accountable history.

Temporal behavior is a policy choice. Not every bounded unit needs the same relation to time. The architecture should name the temporal mode instead of allowing applications to assume that all DSUs carry identical history.

Temporal mode	What it means
Immutable unit	One committed state is intended to remain unchanged. Later interpretation may evolve, but the original content identity does not.
Versioned unit	Material changes extend an ordered history, allowing previous states, authority, and provenance to be examined.
Redirecting unit	A stable human or domain-facing name resolves to another accountable unit and records how the target changes.
Versionless local unit	A lightweight object exposes only its current state and deliberately omits version history. This is suitable for simple local or edge state, but it is not evidentially equivalent to a versioned object backed by a nano-ledger or to a programmable micro-ledger.

The VersionLessDSU direction is important because it prevents the architecture from turning history into dogma. A unit that does not claim durable lineage can remain simple. The boundary becomes dangerous only when a versionless object is presented as though it offered the same continuity, dispute resistance, or reconstructability as a versioned one.

The word *sharing* can be misleading if it suggests that every DSU is widely distributed. A DSU can remain entirely local. It becomes a sharing unit because its access and evidence are structured so that it can cross a boundary without losing its identity. The possibility of controlled exchange is built into the object.

Consider a research dataset. In a conventional workflow, the data may live in object storage, access rights in an identity system, transformation logs in a pipeline database, code in a repository, model parameters in another service, and approvals in email. The dataset exists only because an organization keeps these fragments coordinated.

A DSU-oriented design does not necessarily place everything into one physical package. It creates one accountable logical unit. The content may still be

distributed, but the identifiers, commitments, capabilities, and provenance bind the parts into a reconstructable whole.

This is why the DSU should be understood as a **unit of accountable meaning** rather than as an archive format.

A meaningful boundary changes what the architecture can make explicit.

Consequence	What the boundary enables
Versioning	A material change creates a state that can be compared with, and cryptographically related to, its predecessor.
Attachable authority	Capabilities can apply to the whole unit, a mounted path, a limited operation, or verification alone.
Local provenance	Lineage can remain scoped to the object whose meaning it explains instead of disappearing into one universal audit database.
Composable validation	A unit can declare the rules that govern its transitions without forcing those rules into a global ledger.
Portability	A receiving environment can understand the minimum identity, structure, authority, and evidence without depending permanently on the originating application.

Technical point	Explanation
Logical unity of a DSU	A DSU may be physically distributed across storage services and ledgers. Its unity is logical and cryptographic: the parts can be resolved, verified, and reconstructed as one accountable object.

There is also a social reason to prefer bounded units. Responsibility disappears when systems are too large.

A global ledger asks every participant to accept a common ordering and a common technical surface. A giant enterprise database asks every department to trust one administrative center. A platform data lake collects everything because

future uses are unknown. In each case, the scale of the container exceeds the scale at which people can meaningfully understand consent, purpose, and consequence.

Per-object or per-process units allow governance to become more granular. A clinical consent can be attached to the relevant data unit. A supply-chain event can be verified without exposing unrelated commercial data. A model run can carry its own inputs, parameters, and outputs. A contract negotiation can preserve the sequence of offers without publishing the confidential text to the world.

Granularity does not automatically produce good governance. Millions of badly designed units can create chaos. Naming, discovery, policy inheritance, and lifecycle management remain necessary. But the unit gives governance something precise to operate on.

In the AI age, this precision becomes indispensable. An agent should not receive a broad account credential when it needs one temporary capability to read one dataset for one declared purpose. It should not write to an unbounded enterprise data store when it needs to append one signed result to one accountable history. The DSU boundary makes least authority technically plausible.

CHAPTER 6

Encrypted Bricks, BrickMaps, and Reconstruction

Frame	Chapter focus
Why	Secure collaboration requires confidential content to remain protected even when storage, transport, and application infrastructure are not fully trusted.
How	Content-addressed encrypted bricks, a protected BrickMap, capability-sensitive resolution, and contextual mounting separate custody from interpretation.
What	Authorized environments reconstruct only the permitted view, verify its continuity, and preserve the identity and policy of mounted units.

A secure history is not enough. The content itself must live somewhere, and the place that stores it may be curious, compromised, commercially motivated, or simply outside the owner's control.

OpenDSU's answer was to split a data unit into encrypted, content-addressed pieces called bricks. A special encrypted structure, the BrickMap, records how the pieces form files, folders, or other logical resources and which symmetric keys are needed to reconstruct them.

The implementation details can change. Four durable ideas define the relationship between storage and meaning.

Durable idea	Architectural consequence
Storage without understanding	A provider can retain and replicate encrypted pieces without being trusted with the plaintext.

Durable idea	Architectural consequence
Self-identifying content	A cryptographic digest lets the receiver detect substitution or corruption independently of the store's naming.
Protected structure	The BrickMap or manifest is secured separately because it reveals organization, dependencies, and the keys or references needed for reconstruction.
Authorized reconstruction	Plaintext is materialized only inside an environment where the required capability and policy are present.

This produces a useful reversal of the conventional cloud model. In a conventional application, the server stores structured plaintext and the client asks the server for authorized views. In a brick-based system, the server stores opaque pieces and the client or controlled agent reconstructs the authorized view.

This design shifts interpretation and final integrity verification from the storage provider to the client or another controlled agent.

The security benefit is not that encrypted storage is novel. Encryption at rest is common. The difference is architectural. In many cloud systems, the provider also controls the key-management service, the application logic, the identity layer, and the audit log. Encryption protects the disk while leaving the provider inside the trust boundary. In a DSU-oriented design, the content keys and reconstruction logic can remain under a different security context.

Confidentiality therefore does not require the storage provider to become the permanent authority over interpretation, access, and integrity.

The separation between protected content and public commitments becomes concrete here. Bricks contain the protected data. The anchor or nano-ledger contains compact continuity commitments. The BrickMap supplies the protected structure and key references required for authorized reconstruction.

Content addressing adds more than integrity. It enables deduplication, caching, replication, and lazy loading. A resolver does not need to download an entire archive before using one file. It can retrieve the BrickMap, verify the relevant references, and fetch only the necessary pieces. This matters for large scientific datasets, media collections, digital twins, and mobile wallets.

The same property supports resilience. Because a brick is identified by its digest, identical encrypted envelopes can be stored in multiple places. A resolver can query several stores and accept any copy that matches the expected hash. Storage migration no longer requires changing the logical identity of the data unit.

Content-addressed encryption does not remove metadata, retention, key-concentration, or canonicalization risks. These concerns should be explicit design inputs rather than treated as implementation details.

Design concern	Required response
Metadata leakage	Traffic patterns, object sizes, update frequency, domain names, access times, and shared references can reveal sensitive activity even when plaintext is protected. Batching, padding, private routing, delayed public witnessing, scoped identifiers, and access-pattern controls may be required.
Retention and deletion	Historical bricks support reconstruction and audit but can conflict with retention limits and deletion rights. Implementations should distinguish physical deletion, logical removal, cryptographic erasure, retained commitments, and garbage collection.
BrickMap concentration	The protected map may contain the relationships and keys needed to reconstruct the complete unit. Highly sensitive units may require layered maps, path-specific capabilities, compartmentalized keys, and explicit key epochs.
Canonicalization	Hashes commit to bytes, not abstract meaning. Policies, provenance payloads, events, and structured data need declared canonical representations so that independent verifiers commit to and validate the same object.

These limitations do not weaken the model. They show why the model is more than "encrypt files and put hashes on a ledger." The security comes from the relationship between encrypted pieces, protected structure, capability resolution, reconstruction, and anchored history.

LightDSU makes the relationship easier to see. Its first version can use a complete BrickMap snapshot for each commit, local content-addressed storage, and ordinary authenticated encryption. This is less optimized than sophisticated diff strategies, but it is conceptually clean. Each committed version has one encrypted map that fully describes the reconstructable state.

That simplicity has strategic value. A technology intended to become infrastructure must have a small comprehensible core. Optimization can follow after the invariants are tested.

Most digital content is treated as inert. An application opens a file, parses a response, or reads a database record. The context that makes the content trustworthy remains outside the object, embedded in the application and infrastructure.

OpenDSU reverses this relationship. A DSU is **reconstructed** from encrypted pieces after the resolver has interpreted its capability, discovered the relevant domain, obtained the anchor history, verified the commitments, and loaded the required BrickMap. The Yellow Book compares DSU reconstruction with a secure boot sequence. The technical point is that reconstruction is not a simple download. It is an ordered verification procedure in which the resolver establishes accepted state, capability, protected structure, validation context, and dependencies before materializing plaintext. The system therefore evaluates a compact reconstruction contract rather than asking only whether the bytes can be retrieved.

Reconstruction check	Question answered
Accepted state	Which version or branch is current under the selected witnesses and freshness policy?
Capability	Does the presented authority permit this reconstruction, view, or operation?
Content integrity	Do the encrypted pieces match the commitments in the protected structure?
Protected map	Which paths, keys, dependencies, and resources define the authorized representation?
Validation context	Which rules, provenance profiles, and policy versions must be satisfied?

Reconstruction check	Question answered
Mounted dependencies	Which related units must be resolved, and which of them may remain hidden or verification-only?

The result is a materialized view whose content remains connected to the context required for its interpretation. Mounting extends this procedure to compositions of independently governed units. A DSU can expose another DSU as a path inside its own logical filesystem. The mounted unit keeps its own identifier, history, keys, and policy while participating in a larger composition.

This produces a modular information architecture. A digital wallet can mount separate units for identity credentials, medical records, tickets, contracts, and applications. A scientific package can mount immutable reference data, a versioned dataset, a code environment, and the outputs of an analysis. An enterprise process can compose a purchase order, shipment record, quality certificate, and payment evidence without merging all content into one database.

Composition matters because sovereignty is easily lost at integration boundaries. Conventional integration often requires copying data into a central schema. Once copied, the original policy and provenance weaken. Mounting allows an application to work with a unified view while the components preserve their own authority.

In contextual mounting, the most useful mounted view is not always a complete view. Contextual mounting resolves each dependency according to the keys, capabilities, persona, and policy present in the current Security Context. One participant may reconstruct a clinical DSU; another may see only that the mount exists and that its history verifies; a third may execute application code without receiving the underlying data.

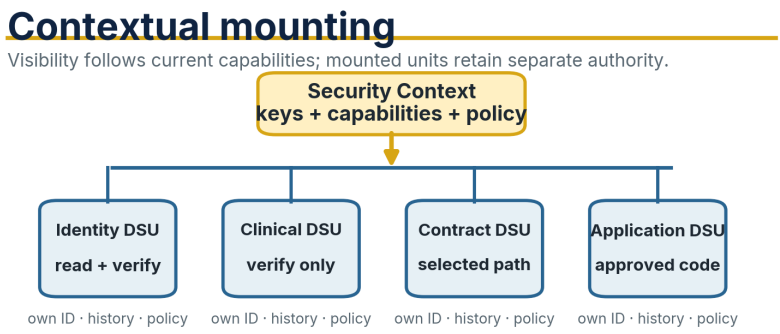

Figure 4. Contextual mounting composes DSUs under one Security Context without merging their identity or authority.

The composition remains stable while visibility changes. This avoids maintaining separate copied packages for every audience and reduces the risk that a derived copy loses revocation, provenance, or policy. A locked mount can remain a meaningful placeholder until an authorized capability enters the context.

Contextual mounting also creates obligations. The verification report must distinguish absent content from unauthorized content, unresolved dependencies from failed validation, and version mismatch from deliberate selective disclosure.

This is not free. A composed system must handle version compatibility, dependency availability, circular references, policy conflicts, and cross-unit consistency. If two mounted units must change atomically, a simple sequence of independent commits may expose an inconsistent intermediate state. The architecture needs explicit choreography or aggregate commitments.

OpenDSU's earlier work on executable choreographies and self-validating data points toward one answer: operations across units can produce a higher-level evidence event that commits to all participating states. Each unit remains locally sovereign, while the aggregate event proves that a coordinated transition was intended.

This idea is particularly useful for agentic systems. A workflow may involve several agents, each controlling different data and tools. Instead of one omnipotent orchestrator, the workflow can be represented as a choreography of signed state

transitions. Each agent validates what it is entitled to see, performs its part, and produces evidence that can be composed into the wider process.

Reconstruction need not reveal the whole unit. A capability may expose only one mounted path, one credential, or one derived view. The BrickMap can be layered so that the resolver sees only the structure needed for the authorized scope.

This is more powerful than encrypting one archive with one password. It allows the information environment to be assembled differently for different principals while preserving a common anchored identity.

Reconstruction also supports recovery because the logical unit is independent from one storage provider. If bricks are replicated and the anchor history remains available, the unit can be rebuilt after a service failure. The wallet or agent becomes portable not because it copies a proprietary database, but because it possesses the capabilities and verifiable references needed to reconstitute its state.

Technical point	Explanation
Data should support independent contextual verification	The strongest future data object will not merely present a value. It will make it possible to reconstruct the value, identify the applicable history, locate the authority, and explain which validation rules were used.

CHAPTER 7

The Anchor as a nano-ledger

Frame	Chapter focus
Why	Treating an anchor as a single hash obscures its role in version continuity, current control, freshness, and branch detection.
How	A stable AnchorID identifies a signed append-only sequence of version commitments and control transitions, optionally witnessed by external infrastructure.
What	The anchor is the nano-ledger of the DSU; anchoring is the operation that extends or externally witnesses that nano-ledger.

OpenDSU uses the term **anchor** for the object-scale history associated with a DSU. In this book, **anchor** and **nano-ledger** name the same conceptual structure. The term nano-ledger emphasizes that the anchor is not merely a single hash or a pointer stored somewhere else. It is a compact ledger identified by an AnchorID and extended through cryptographically authorized append operations.

The simplest nano-ledger has one recurring operation: append a commitment to a new DSU version. Each entry binds the stable AnchorID, the preceding entry, a timestamp or sequence claim, and the hash link of the BrickMap representing the new version. The controller signs the operation. The resulting sequence is independently verifiable even when the confidential BrickMaps and data bricks remain outside the anchoring ledger.

This definition separates four concepts that are frequently confused.

Concept	Precise role
Anchor or nano-ledger	The compact per-object append-only history of signed version commitments and permitted control transitions.

Concept	Precise role
AnchorID	The stable identifier of that nano-ledger, normally related to the public key or control information used to validate appends.
Anchoring operation	The act of extending the nano-ledger with a new authorized entry or registering a checkpoint of it with an external witness.
Anchoring service or ledger	Infrastructure that stores, orders, replicates, or witnesses nano-ledger entries and helps a verifier establish currentness and detect equivocation.

The distinction matters because an anchor can exist before a distributed ledger is introduced. A local implementation may store the signed sequence in an append-only file. A company may place it in an organizational transparency service. A consortium may register it in a permissioned ledger. A public infrastructure may receive selected checkpoints. These choices change the strength and independence of the witness, but they do not change the identity of the nano-ledger.

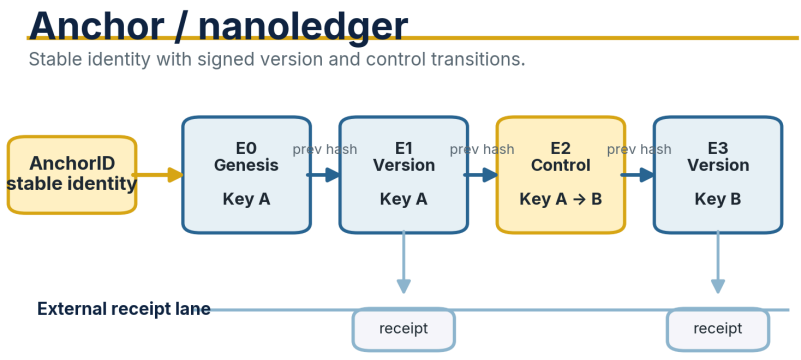

Figure 5. The anchor is the nano-ledger of one DSU: a stable identity with signed version commitments and control transitions.

The Axiologic tokenisation report describes the anchor as a minimalistic smart contract. The comparison is useful if interpreted structurally rather than as a claim about execution technology. A conventional smart contract can expose many

operations and maintain globally agreed state. A nano-ledger can remain far smaller: it admits only the transition types required to preserve object continuity, normally a version append and, when supported, a control or ownership transition.

A control transition illustrates why the anchor is a ledger rather than a static pointer. The object identity should remain stable while the authority to append future entries changes. The current controller signs an entry containing the new public control key. Subsequent version commitments are accepted only under the new key. The prior controller remains visible in history, but cannot legitimately extend the object after the transition.

This mechanism supports a precise interpretation of ownership for unique digital objects. Ownership is not inferred from possession of bytes, because encrypted content can be copied. It is represented by the current authority to admit successors to the stable nano-ledger, together with whatever legal or domain rules give that authority meaning. A transfer therefore changes control of future history rather than rewriting the object identity.

Object operation	nano-ledger interpretation
Create or mint	Establish an AnchorID, initial controller, first content commitment, and the domain in which the object will be resolved.
Update metadata or content	Append a signed commitment to a new BrickMap or object state while preserving the prior entries.
Query current controller	Evaluate the latest valid control transition under the accepted branch and witness policy.
Transfer control	Append an authorized transition to a new control key while retaining the same AnchorID and past sequence.
Retire or burn	Append a terminal policy or control state and, when policy permits, delete or cryptographically erase the protected content while retaining a minimal audit commitment.
Delegate operation	Derive or register a constrained capability that permits a limited operation without transferring full control of the nano-ledger.

The comparison with non-fungible tokens also clarifies where the model is not appropriate. A DSU and its nano-ledger are natural for unique or bounded objects whose content, metadata, provenance, and control history matter. Fungible assets that require atomic global balance changes may still require a consensus system with shared execution. Object-scale ledgers do not remove the need for global consensus where the domain genuinely contains global invariants.

The external ledger has a narrower responsibility than is often assumed. The nano-ledger entries already use signatures and chaining to make unauthorized modification detectable. A shared anchoring ledger primarily strengthens **freshness**, **single-branch visibility**, and **availability**. It helps the verifier determine that the history presented is the latest accepted history rather than a valid but truncated prefix or a branch shown only to one participant.

Freshness should not be reduced to one timestamp. A verifier needs to know the last sequence known to the required witnesses, whether a pending append exists, whether an authority epoch has changed, and what delay the domain permits between local commit and external receipt. An old entry can remain perfectly authentic while being unsafe for present use.

The nano-ledger is deliberately compact. Detailed provenance, validation traces, prompts, model outputs, business documents, and personal data should normally remain encrypted and be referenced by commitment. This protects confidentiality and keeps verification costs proportional. A compact history can still become large in high-frequency systems, so aggregation, Merkle checkpoints, retention rules, and evidence summaries may be required.

nano-ledgers can be related without being merged. A transfer between two DSUs can produce an aggregate event that commits to the accepted state of each nano-ledger. A ledger can checkpoint a batch of nano-ledger roots. A micro-ledger can record a complex domain process and place its accepted state commitment into the DSU anchor. These compositions preserve local object identity while allowing higher-level coordination.

The anchor-as-nano-ledger definition corrects a recurring conceptual error. The nano-ledger is not a future refinement smaller than the anchor. It is the anchor understood as the minimal cryptographic ledger of one object. The term is useful because it makes clear that OpenDSU distributes ledger properties to the scale at which continuity and authority are actually needed.

CHAPTER 8

micro-ledgers and Self-Validating Data

Frame	Chapter focus
Why	Version commitments alone cannot establish whether complex domain operations produced logically valid successor states.
How	A scoped micro-ledger records domain commands and validation semantics; an SVD implements this by binding history, replay, provenance, and rules to the data.
What	The DSU may store the complete SVD micro-ledger off-chain while its anchor or nano-ledger commits accepted states for freshness and shared verification.

A nano-ledger answers a narrow question: which commitments and control transitions form the accepted continuity of one bounded object? Many applications require a richer question: did each domain-specific operation produce a valid successor state according to the rules that governed the object or process? OpenDSU uses the concept of a **micro-ledger** for this richer scoped history.

A micro-ledger is a ledger at the scale of one data structure, process, relationship, or smart-contract-like object. It can support many operation types, validation rules, actors, and state transitions without requiring a global consensus network to execute every rule. The history remains bounded, and external witnesses can be used only for the ordering and independence that the threat model requires.

Self-Validating Data is a possible implementation of a micro-ledger. An SVD combines a history representation, provenance metadata, primitive cryptographic operations, state operations, and replay or validation logic. A receiving environment loads the history, associates the applicable validation artifacts, replays the accepted commands, and obtains a state that it can read or extend only if validation succeeds.

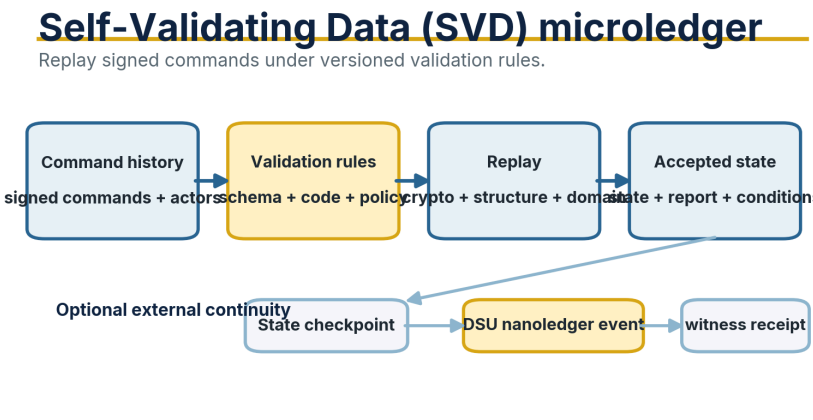

Figure 6. Self-Validating Data is a possible micro-ledger implementation based on signed history, replay, and domain validation.

This relation should be stated carefully. Not every micro-ledger must be an SVD. A micro-ledger may be implemented through a conventional event-sourced service, a domain-specific ledger, a replicated state machine, a DAG, or a CRDT with appropriate validation semantics. Not every DSU must contain a micro-ledger. A DSU may be an immutable package, a versioned content object, a wallet, or a versionless local unit. SVD is the pattern used when the history and the rules for interpreting that history must remain bound to the data.

The research article on SVD identifies four principal properties.

SVD property	micro-ledger consequence
Cryptographic provenance	Material changes are attributable through digital signatures or equivalent actor evidence, allowing responsibility for state transitions to be evaluated.
History integrity	Commands or state commitments are hash-linked so that alteration, deletion, or reordering of accepted past entries becomes detectable.
Self-validation	Manipulation rules are defined before a transition is accepted, allowing stakeholders to evaluate logical correctness rather than only signature validity.

SVD property	micro-ledger consequence
Composability	Several independently governed SVDs can participate in an aggregate transition whose evidence commits to all relevant states without dissolving their separate histories.

The phrase self-validating does not imply that data executes itself or guarantees truth. A compatible execution environment performs the validation. The term describes architectural locality: the object carries or resolves to the history, rules, credentials, provenance, and execution references needed for the receiving environment to evaluate the transition without relying exclusively on a remote service controlled by the originator.

An SVD separates primitive operations from state operations. Primitive operations provide stable facilities such as hashing, signature verification, key resolution, time or witness access, and interaction with the execution environment. State operations define how domain commands read or transform the current state. This separation allows different language implementations to agree on semantics while using different runtimes.

Layer	Responsibility
History representation	Preserves commands, dependencies, sequence, causal relations, or conflict information using a chain, DAG, CRDT, or another declared model.
Primitive operations	Provide deterministic cryptographic and environmental functions that the validator can invoke consistently.
State operations	Define the domain-specific commands and permitted transformations of the data structure.
Validation artifacts	Bind schemas, policies, executable modules, credentials, and external references to the version under evaluation.
Replay and state derivation	Apply the accepted history to reconstruct a state and produce a validation report that identifies failures and unresolved dependencies.

Layer	Responsibility
External anchoring	Commit selected states or history roots to the DSU nano-ledger or another witness so that participants can detect truncation, forks, or later rewriting.

The most straightforward SVD implementation records a command for each change and derives the current state by replay. This model provides complete interpretability but may become expensive as the history grows. Checkpoints, snapshots, proofs of replay, compact state commitments, and selective validation can improve performance, provided that the historical rules and the path from checkpoint to current state remain verifiable.

The representation of history is a design choice. A linear chain provides simple ordering and is appropriate when one accepted successor must be selected. A DAG preserves parallel contributions and causal relations, but requires conflict and merge rules. A CRDT supports convergence among replicas, but the domain still needs to define which operations are authorized and which converged states are acceptable. The essential requirement is that the representation and validation semantics are explicit.

A **micro-ledger SVD** can keep its full command history off-chain in a DSU while the DSU nano-ledger stores commitments to accepted micro-ledger states. This division is important. The micro-ledger contains the rich programmable history; the nano-ledger provides compact continuity and can receive external witness receipts. The shared ledger does not need to execute confidential business rules or store the underlying data.

This architecture supports the earlier OpenDSU concept of secret smart contracts. The phrase describes placement, not a claim that hidden code is automatically trustworthy. Domain logic executes near confidential data inside an authorized Security Context. The witness records commitments and order. Participants can independently replay or validate the micro-ledger when they possess the required capability and artifacts.

Executable choreographies are a natural application. Instead of one central service orchestrating every participant, each organization or agent validates the input state it is authorized to see, performs its permitted transition, and returns signed evidence. An aggregate event can commit to the participating micro-ledger

states. This model can reduce unnecessary exposure and central control, but it requires clear failure handling, deterministic interfaces, bounded side effects, and portable evidence.

AI workflows make these requirements more concrete. A task package can identify the input DSU versions, permitted tools, model or service references, policy, expected output commitments, and required provenance profiles. The receiving agent executes inside a constrained Security Context. The result becomes a command in the relevant micro-ledger only when the required evidence is present. The agent's natural-language explanation may accompany the event, but it does not replace the committed inputs, authority, and validation report.

The SVD idea also applies to collaborative knowledge. A scientific claim, review, correction, and competing interpretation can be represented as attributable operations rather than overwritten text. A multi-perspective publication need not force all participants into one consensus view; it can preserve claims, dependencies, reviews, and audience-specific projections while maintaining verifiable authorship and history. The challenge is not only cryptographic. Relevance, moderation, incentives, privacy, and lawful removal require governance.

SVD therefore occupies a precise place in the framework. The DSU provides the protected object boundary and reconstructable storage. The anchor or nano-ledger provides compact continuity and current control. The micro-ledger provides domain-specific state history. Self-Validating Data is one method for implementing that micro-ledger so that rules and evidence remain available where the data is interpreted.

CHAPTER 9

KeySSI and Capability-Based Authority

Frame	Chapter focus
Why	Broad accounts and copied master secrets give autonomous systems more power than a specific task requires and make authority difficult to audit.
How	KeySSI families combine identification, cryptographic material, derivation, attenuation, and domain resolution while separating control, read, and verification-only authority.
What	A recipient can receive the narrow capability needed for one object or operation without depending permanently on the originating platform account.

Most digital identifiers answer one question: what object or account are we talking about? Access tokens answer another: what may the bearer do? Cryptographic keys answer a third: who can sign or decrypt? In conventional systems these functions are often separated across URLs, usernames, passwords, access-control databases, certificates, and key-management services.

The KeySSI intuition is to bring them into one coherent capability system.

A KeySSI is not valuable because it begins with a particular string or because one implementation encodes fields in a certain order. Its conceptual value is the union of three roles.

Role	What the KeySSI pattern contributes
Identity	It identifies a data unit or the trust domain in which the unit can be resolved.
Authority	It carries or resolves the cryptographic material that permits control, reconstruction, or verification.

Role	What the KeySSI pattern contributes
Attenuation	It expresses an access level that can be derived, delegated, narrowed, and independently checked.

This is stronger than a conventional identifier and safer than a universal account credential.

The early OpenDSU families illustrated the idea through a hierarchy. A high-power key could control and update a DSU. From it, a read capability could be derived. From the read capability, a zero-access identifier could be derived that revealed only enough to verify the anchor and version history. LightDSU preserves the same functional structure with three compact roles: full control, read and reconstruction, and verification without content access.

The terminology may change, but the separation among control, read, and verification-only authority should remain invariant.

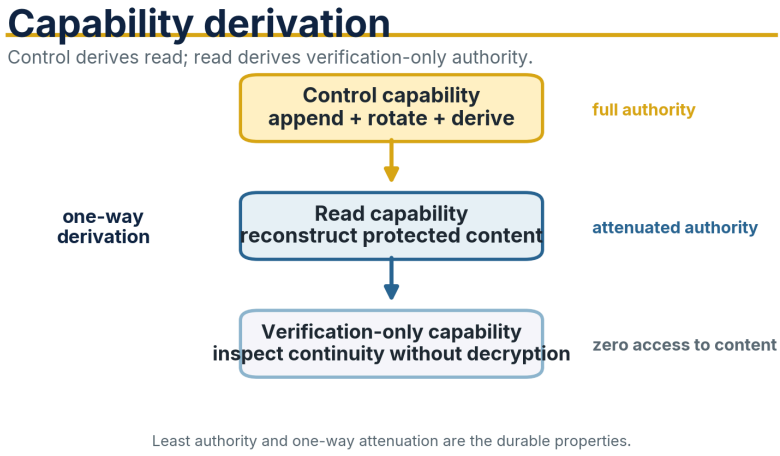

Figure 7. A KeySSI family attenuates authority from control to reading and verification-only access.

Zero access is not zero knowledge. OpenDSU's *zero-access* idea and cryptographic *zero-knowledge proofs* solve different problems. Conflating them makes architectures sound stronger than they are.

Mechanism	What it actually provides
Zero-access capability	Permits a holder or infrastructure service to identify an anchor, inspect continuity, or verify selected commitments without possessing the keys needed to reconstruct protected content.
Zero-knowledge proof	Lets a prover demonstrate that a statement is true under a defined circuit or relation without revealing the private witness used to prove it.
Encryption	Keeps plaintext unavailable to parties without the decryption key, but does not by itself prove arbitrary facts about the hidden content.
Selective disclosure	Reveals a deliberately limited claim or view while withholding unrelated attributes or mounted content.

A zero-access witness can preserve history without reading the object. A zero-knowledge system can prove a specific property of hidden data. Either may complement the other, but neither should be advertised as proof that the underlying statement is factually true.

The pattern also becomes clearer when it is distinguished from adjacent identity concepts.

Instrument	What it is responsible for
KeySSI or capability	Expresses a concrete cryptographic relationship to a unit: control it, reconstruct it, invoke a bounded operation, or verify continuity.
DID or identity document	Provides a resolvable identifier and public verification material for an actor, organization, device, or service.
Verifiable Credential	Carries an issuer's signed claim about a subject, role, qualification, status, or permission.

Instrument	What it is responsible for
Persona or profile	Selects a portable operational view, preferences, mounted units, and expected application context without replacing the underlying identity or capability.
Security Context	Holds keys and policy, decides which capability may be used, and performs sensitive operations without exporting raw secrets.

Collapsing these instruments creates dangerous ambiguity. A credential that says an agent is a purchasing agent does not itself grant access to every purchase-order DSU. A DID identifies the actor but does not establish current authorization. A capability authorizes an operation but does not prove legal personhood. A profile or persona organizes context but should never become an invisible source of ambient authority.

This hierarchy embodies **least authority**. A person who needs to verify that a record exists should not receive the power to decrypt it. A service that needs to read a dataset should not receive the power to publish a new version. An AI agent that needs one temporary input should not inherit the full authority of the human or organization that launched it.

Derivation creates a one-way delegation path. The holder of a stronger capability can produce a weaker one, but the weaker capability cannot recover the stronger secret. This is conceptually different from asking a central access-control server to remember an entry for every delegation. The cryptographic relationship carries part of the proof.

This does not eliminate central policy. Organizations may still require directories, role systems, approvals, and revocation registries. But the capability can remain meaningful outside the original application. It is portable evidence of a specific authority relationship.

The combination of identifier and capability has another advantage: it narrows the gap between naming and security.

On the ordinary web, a URL identifies a location and TLS authenticates a server, but the right to read the resource is usually mediated through a session, cookie, account, or bearer token. The user must trust the server to apply the right policy and serve the right version. In a KeySSI-oriented system, the identifier can

carry enough information for a resolver to locate the relevant trust domain, verify the anchor, and derive the decryption context locally.

The storage service is reduced to a utility. It may still deny service, observe traffic, or return corrupt data, but it need not be trusted to understand plaintext or make the final integrity decision.

This reduction in delegated trust is central to Data Sovereignty.

Capability systems also create operational risks. A secret capability can be copied; a human-readable identifier derived from weak input can be guessed; a long-lived master capability can become a single point of catastrophic loss; and an AI agent can leak authority into prompts, logs, tool calls, or memory. The architecture therefore needs explicit capability-lifecycle disciplines.

Capability discipline	Required property
Separate control from operation	The key that controls long-term DSU history should not be used automatically by every operational actor. LightDSU's separate anchor and actor signatures distinguish inclusion in object history from responsibility for the semantic action.
Derive narrow and temporary authority	Capabilities should be scoped by object, operation, purpose, path, and time whenever risk justifies it. Delegation should attenuate authority rather than copy full control.
Record revocation and authority epochs	Revocations, key epochs, policy changes, and recovery transitions should be part of the same verifiable history. A resolver must establish whether authority was valid at the relevant sequence and time, not merely whether a signature verifies.
Preserve historical interpretation	Archived signatures remain interpretable under their historical authority epoch, while new actions must satisfy the current key and delegation state.
Support cryptographic agility	Identifiers and evidence formats should permit migration of hash, signature, derivation, and encoding conventions without replacing object identity.

Capability discipline	Required property
Keep identity distinct from keys	A person, organization, device, or agent is not reducible to one key. Signatures prove control of cryptographic material; governance, recovery, credentials, and legal context establish broader identity and responsibility.

Technical point	Explanation
Capability semantics	A well-designed capability says something precise: "The bearer may perform this operation on this unit under this policy." A badly designed credential says only: "The bearer is inside." The constrained form is preferable in systems that require least authority and independent verification.

The rise of autonomous agents makes capability design a foundational security problem. An agent may be competent enough to perform a task and still be unsafe if its authority is too broad. It may be hijacked, confused by an adversarial input, or induced to use a legitimate tool for an illegitimate purpose. The correct response is not to trust the agent's reasoning more. It is to give the agent authority that is narrow enough to survive imperfect reasoning.

This is where KeySSI becomes more than an OpenDSU mechanism. It becomes a pattern for **agentic authorization**.

An agent should be able to present a cryptographically verifiable capability tied to a data unit, purpose, policy, and evidence trail. The receiving system should be able to verify the capability without opening an unrestricted session. The resulting action should be appended to the unit's history under both the object's control key and the actor's identity.

CHAPTER 10

Security Contexts, Wallets, and Self-Sovereign Applications

Frame	Chapter focus
Why	Control of keys is ineffective when arbitrary application code can invoke or export them without policy, user intent, or attributable evidence.
How	A Security Context places keys, identities, enclaves, capabilities, mounted DSUs, and execution policy inside one controlled authority environment.
What	Wallets and Self-Sovereign Applications become portable, verifiable contexts for using data rather than permanent owners of that data.

The word *wallet* entered computing through cryptocurrency, but OpenDSU uses it in a broader sense. A wallet is a software environment under an agent's control that manages cryptographic authority and private data.

This definition matters because keys without an execution environment are unusable. A person cannot meaningfully control a capability if every application can silently extract it. An enterprise cannot claim sovereignty if private keys travel through generic microservices and application logs. A secure system needs a place in which sensitive operations occur without exposing the secrets themselves.

OpenDSU called that place a **Security Context**, often implemented through an **enclave**.

The enclave may be local software, a mobile secure element, a trusted execution environment, a hardware security module, or a remote key-management system. The implementation is secondary. The essential rule is that private or secret keys

should not leave the controlled boundary. Signing, decryption, and key derivation happen inside. Applications request an operation, not the raw secret.

This prevents a common antipattern: treating a key-management system as a decryption web service that hands plaintext to any application with network permission. The key is protected, but the authority is still ambient. A better enclave evaluates both the caller and the intended operation.

Wallet architecture also clarifies the edge-cloud debate.

An **edge wallet** maximizes personal or local sovereignty. Keys and reconstruction happen on a device controlled by the user or organization. The advantages are privacy, independence, and reduced server visibility. The disadvantages are device loss, limited availability, constrained computation, and difficult recovery.

A **cloud wallet** maximizes availability, integration, and organizational management. It can run continuously, connect to enterprise systems, and use professional backup and hardware security. The disadvantages are concentrated trust, metadata exposure, and the risk that the service provider becomes an unchallengeable intermediary.

The correct architecture is often mixed. A high-value signing key can remain in a hardware enclave while encrypted bricks are replicated in the cloud. A personal device can approve sensitive operations while a cloud agent handles notifications and availability. An enterprise can maintain a shared enclave for organizational data while preserving separate personal contexts for employees or patients.

This illustrates the OpenDSU principle that no single deployment topology fits every security, availability, and governance context.

Application sovereignty is required because data sovereignty remains incomplete when the application that interprets the data is controlled elsewhere.

A web application can present a user with "their" data while the server decides which fields exist, which history is visible, which algorithms run, and which exports are permitted. The user owns an interface, not the computational relationship.

The Self-Sovereign Application, or SSApp, extends sovereignty to code. Data and signed application logic can be packaged in DSUs and executed in compatible wallets or sandboxed environments. The application becomes portable with the data rather than permanently attached to one server.

This does not imply that every application should be fully serverless. Servers remain valuable for availability, search, aggregation, collaboration, and heavy computation. The philosophical change is that servers become replaceable services rather than the sole home of identity and meaning.

A Self-Sovereign Application can be evaluated through four architectural conditions.

Condition	Design test
Controlled authority	Does the user or governing organization control the Security Context in which keys and capabilities are interpreted?
Verifiable code	Can the environment identify the code, rules, model, or application version that interprets sensitive data?
Portable state	Can the data and its evidence be moved or reconstructed without the original application provider?
Least disclosure	Do external services receive only the information and authority required for their declared role?

Portable behavior without a permanent endpoint. The conceptual contribution of DSU APIs is that behavior can become available after an authorized unit is reconstructed. The interface belongs to the portable object and its constitution rather than only to a permanent server endpoint.

Interaction model	Where authority and verification reside
Conventional Web API	The provider owns the endpoint, mediates every invocation, and usually controls the account, current state, and interpretation of the response.
Reconstructed DSU interface	The receiving wallet or agent resolves the capability, validates the unit, and exposes only the operations defined for that verified state inside its Security Context.

Interaction model	Where authority and verification reside
Hybrid adapter	A conventional service performs search, computation, messaging, or integration, while the resulting state transition and evidence return to the accountable unit.

This does not eliminate networks or enterprise services. It makes them replaceable participants in a protocol whose state, authority, and evidence can survive the disappearance of one endpoint.

Identity, persona, capability, and context. A sovereign wallet should not reduce every interaction to one permanent identity. It needs a vocabulary for choosing an operational view without confusing presentation with authority.

Concept	Function in a sovereign wallet
Identity	Establishes the actor or organization that can be held accountable across contexts.
Credential	Provides a signed claim about status, role, qualification, or relationship.
Persona or profile	Selects a portable view of mounted units, preferences, applications, and disclosure expectations for a situation.
Capability	Grants precise authority over a particular unit or operation.
Security Context	Combines the active identity, capabilities, policy, enclave, and verification state from which an operation is evaluated.

The OpenDSU design direction around DSU Profiles and personas is valuable because it separates a user's or agent's portable working context from one monolithic wallet. It remains a design direction rather than a reason to make every profile authoritative. The profile may select what is visible; only explicit capabilities should decide what may be done.

The same model is directly relevant to AI assistants because their authority should be constrained by the Security Context in which data, tools, and capabilities are resolved.

A personal AI agent will need access to calendars, health records, contracts, messages, credentials, and financial information. If the agent exists only as a cloud account controlled by one platform, it becomes the most powerful intermediary ever created. It can observe the user's whole life and act across domains.

A wallet-based agent can be designed differently. The security context holds the user's capabilities. The agent receives temporary, purpose-limited access. Sensitive data can be reconstructed locally. Tool calls can be signed and appended to relevant verifiable event chains. A different model can replace the reasoning engine without transferring ownership of the user's entire history.

This separation between the **reasoning provider** and the **authority holder** may become one of the most important constitutional boundaries of the AI age.

Technical point	Explanation
The wallet is an authority context	A wallet is not merely a container of secrets. It is the place where authority is interpreted. It decides which code may request an operation, which capability applies, what evidence must be produced, and when a human or independent policy must intervene.

Self-Sovereign Applications also improve ecosystem interoperability. If code, data, and capabilities follow open reconstruction and execution conventions, different wallets can host the same functional unit. An organization can replace the shell without losing the application. A person can move an application from one device to another while preserving its anchored history.

The goal is not to make all software identical. It is to make the boundary between applications and sovereign data explicit enough that innovation does not require captivity.

CHAPTER 11

BDNS and Plural Trust Domains

Frame	Chapter focus
Why	Real digital ecosystems use heterogeneous ledgers, stores, jurisdictions, witness policies, and organizational roots rather than one universal infrastructure.
How	BDNS associates an identifier domain with verifiable configuration for storage, anchoring, resolution, validation, and migration.
What	A DSU capability remains meaningful across plural services while the relying party can evaluate the trust policy of the selected domain.

A technology becomes brittle when it assumes that one infrastructure will rule forever.

The early blockchain imagination often made this assumption explicitly. One universal chain would contain the authoritative state of assets, identities, contracts, and organizations. The assumption simplified diagrams and complicated reality.

Different systems have different needs. A national identity registry, a factory ledger, a personal health wallet, a public transparency service, and a laboratory instrument do not need the same consensus model, privacy boundary, governance, latency, or cost. Some histories should be witnessed by a public network. Others should never reveal even their update frequency outside a small group.

OpenDSU responded with hierarchical and heterogeneous ledgers connected through the Blockchain Domain Name System, or BDNS.

At the implementation level, BDNS maps a ledger domain to anchoring services, brick stores, notification endpoints, and other configuration. At the philosophical level, it solves a larger problem: how can an identifier remain meaningful in a plural world of trust infrastructures?

The ordinary Domain Name System maps human names to network locations. BDNS adds a trust dimension. The domain can indicate which ledger, witness set, storage topology, and verification policy belong to a data unit.

This function can be described precisely as **verifiable trust routing**: resolution selects both infrastructure endpoints and the rules under which their results may be trusted.

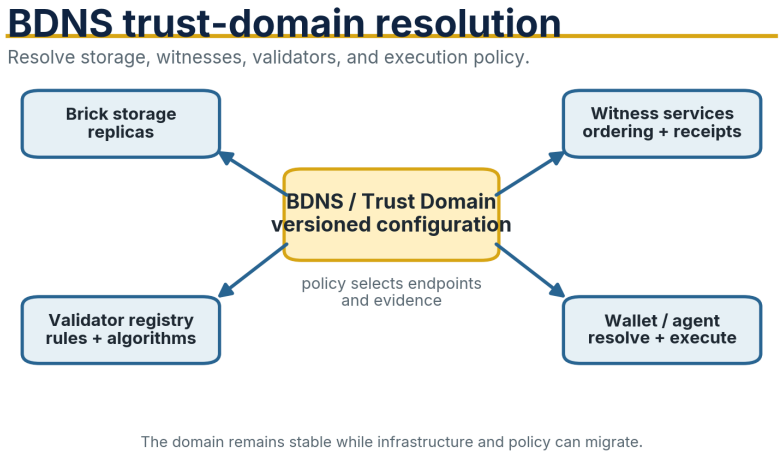

Figure 8. A trust domain resolves heterogeneous infrastructure, historical configuration, policy, and witnesses.

A resolver should not merely ask, "Where is the server?" It should interpret the trust domain as verifiable configuration.

Resolver question	Why it matters
Governing domain	Identifies the policy, jurisdiction, roots, and naming authority that give the identifier operational meaning.
Witness set	Shows where the anchor history can be checked and how freshness or consistency is established.
Storage topology	Locates permitted replicas of the encrypted bricks without making any one store authoritative for plaintext or integrity.
Cryptographic version	Declares the suites and conventions required to interpret identifiers, events, and receipts.

Resolver question	Why it matters
Verification depth	States how many gateways or independent witnesses a relying party should query for the intended consequence.
Parent checkpoint	Identifies a higher-level or public commitment that strengthens a local or consortium history.
Fallback and succession	Defines how the unit remains resolvable when a gateway, witness, or domain operator disappears.

The answer may be a hierarchy. A company domain can anchor into a consortium domain, which periodically anchors into an independent public service. A personal wallet can keep most events local and use a public timestamp only for selected commitments. A regulated ecosystem can maintain several national or sector-specific ledgers while sharing common naming and evidence formats.

The value of hierarchy is not that higher is always better. Higher levels usually bring broader independence and greater cost or exposure. Lower levels bring privacy, speed, and operational control. The architecture lets a use case choose the appropriate level and migrate later.

This migration capacity is one of BDNS's most durable insights.

Cryptographic and ledger technologies age. A system may begin with a local append-only log, move to a consortium transparency service, replace one distributed ledger with another, or adopt post-quantum signatures. If identifiers are inseparably bound to one vendor or chain, migration becomes a historical rupture. A naming and trust-routing layer can preserve logical continuity while the substrate changes.

Plurality also supports resilience against compromised gateways.

A lightweight client may not store the complete state of an anchoring ledger. It asks gateways for the current history. If only one gateway exists, that gateway can truncate, delay, or selectively present the history. BDNS can publish multiple gateways and allow the resolver to choose a declared verification depth. A low-risk operation may query one endpoint. A high-risk operation may compare several independent responses and require a matching witness receipt. Verification effort should be proportional to consequence.

BDNS itself needs governance. A naming system can become a central point of capture. Whoever controls the mapping can redirect clients to malicious gateways or obsolete trust roots. Therefore the BDNS configuration must itself be signed, versioned, anchored, and distributed through more than one channel. Root changes should be visible, and clients should preserve trusted checkpoints.

The deeper lesson is that there is no final escape from governance. Decentralization changes where governance happens; it does not remove the need to decide who can update names, rotate roots, resolve disputes, and recover from compromise.

In the AI age, verifiable trust routing becomes even more important. Agents will discover tools, data, models, and other agents dynamically. A simple network address is not enough. The agent must know what authority the endpoint represents, which evidence service witnesses its actions, and whether the response belongs to the expected trust domain.

A malicious tool can mimic a legitimate interface. A poisoned model registry can serve a substituted artifact. A compromised agent can advertise a valid name with an invalid policy. Binding discovery to signed trust-domain configuration reduces the space in which these substitutions can remain invisible.

BDNS should therefore be allowed to evolve beyond its blockchain name. Its essential function is not to promote blockchains. It is to let a plural digital world remain navigable without collapsing back into one unquestioned directory.

CHAPTER 12

Provenance, Policy, and Validation Profiles

Frame	Chapter focus
Why	Integrity alone cannot explain how a dataset, clinical record, model result, or regulated decision was created, transformed, accessed, or approved.
How	Compact provenance events commit encrypted profile payloads, while policy declares which profiles and fields are mandatory for each operation.
What	General, clinical, GxP, research, genomic, enterprise, and AI/ML provenance can share one event relationship without exposing all details publicly.

Provenance is often treated as a descriptive appendix: a few fields showing where a file came from. In future digital systems, provenance must become operational.

The W3C provenance model offers a useful foundation by distinguishing **entities**, **activities**, and **agents**. An entity is the data or artifact. An activity is the process that used or generated it. An agent is the person, organization, device, or software that bears responsibility.

This triad is more powerful than a creation timestamp because it captures transformation. Most important data does not have one origin. It passes through acquisition, normalization, analysis, aggregation, review, publication, and reuse. Each step changes what the data can mean.

OpenDSU's philosophy adds two requirements that must hold together.

Requirement	Meaning
Cryptographic binding	Provenance is committed to the exact version, resource, or action it describes, so it cannot be detached and reused as generic narrative.

Requirement	Meaning
Confidential detail	Rich lineage may remain encrypted while a compact commitment and profile identifier enter the verifiable event chain.

LightDSU's provenance extension makes this pattern explicit. A PROVENANCE event contains only the event type, a commitment to an encrypted payload, an optional resource reference, actor evidence, and signatures. The payload declares a profile and contains the domain-specific detail.

This separation solves an important tension. Auditability needs durable evidence; privacy needs restraint. A public anchor can prove that a provenance object existed and has not changed, while the detailed content is disclosed only to authorized parties.

Profiles are necessary because a generic provenance schema is either too small for serious use or too large for adoption. Profile-based provenance allows the core evidence format to remain stable while different domains define what must be recorded.

Profile families let the same compact event format serve very different evidentiary domains.

Profile family	What it adds
Minimal technical	Operation, actor, input and output commitments, method, software version, time, and reason.
Scientific and reproducibility	Dataset identifiers, random seeds, container images, environmental parameters, evaluator, and metrics.
Clinical and healthcare	FHIR Provenance for origin and transformation, with FHIR AuditEvent for access and operational accountability.
Regulated GxP or GLP	Previous and new values, reason for change, electronic-signature meaning, review state, retention class, system version, and regulated context.

Profile family	What it adds
Research package	RO-Crate metadata for FAIR aggregation, authorship, workflow references, licensing, preservation, and publication.
Genomic data use	Machine-readable consent, permitted purposes, prohibited uses, policy version, and data-access governance.
AI and ML experiment	Model and version, prompt or input commitments, code and container hashes, parameter state, software environment, evaluator, and output commitments.
Enterprise data quality	Source, acquisition method, responsible party, transformation history, validation state, and master-data quality status.

The anchor does not need to know all these fields. It commits to the encrypted payload and identifies the profile.

Provenance must be distinguished from access logging. A mature system distinguishes three event classes.

Event class	Question it answers
Access log	Who used, exported, shared, derived, or analyzed a resource, and under which permission?
Provenance	How did an artifact come to exist or change through import, transformation, normalization, training, evaluation, approval, or revision?
Policy history	Which retention, audit, access, profile, validation, or key-epoch rules governed the unit at each point?

One operation may create several events. An AI analysis of genomic data might produce an access event for reading the dataset, a provenance event for the model run, a data-use record showing that the purpose was permitted, and a policy event if the output enters a regulated retention class.

This is not bureaucratic duplication. The events answer different questions.

A provenance policy functions as a contract by declaring which profiles are required for which operations. The engine should refuse a commit when the required evidence is missing.

This makes provenance a **enforceable evidence policy** between the data unit and every actor that transforms it.

A dataset can state: "Any derived model must include the dataset commitment, code commitment, model version, parameters, evaluator, and output hashes." A regulated record can state: "Any modification must include a reason and an attributable actor." A consent-bound dataset can state: "Any use must record the declared purpose and the applicable policy version."

The contract travels with the data rather than remaining in a compliance manual that the software may ignore.

Technical point	Explanation
Provenance requirements should govern successor states	The strongest provenance system does not merely explain yesterday. It states what evidence tomorrow's transformations must produce before they can be accepted as valid successors.

AI reproducibility creates unusual provenance requirements because AI systems may depend on artifacts and services that are difficult to preserve. A result may depend on a model whose weights are unavailable, a remote API that changes silently, a non-deterministic sampling process, hidden safety filters, retrieval sources that disappear, or prompts assembled dynamically from user data and tool outputs.

Perfect reproduction may be impossible. Honest provenance should therefore record both what is known and what is unavailable.

A model API identifier is not the same as a model-weight hash. A provider's version label is not the same as an immutable artifact. A prompt hash proves commitment to a prompt without necessarily disclosing it. A container hash describes part of the environment but not the external services it called. Provenance profiles should make these distinctions explicit.

This is another reason to avoid the language of automatic truth. Provenance improves the ability to assess a result. It does not guarantee that the result is correct.

Selective disclosure is necessary because detailed lineage can itself be sensitive. It may reveal trade secrets, patient identities, research directions, or security procedures. The encrypted-payload pattern supports several disclosure surfaces.

Disclosure surface	What becomes visible
Public commitment	Only the profile identifier, event relationship, and hash of the encrypted provenance object.
Summary view	A limited set of profile fields chosen for ordinary reliance or public communication.
Auditor view	Detailed lineage required to test compliance, validation, or scientific integrity under controlled access.
Counterparty view	The fields needed for a contractual or cross-organizational decision without exposing unrelated internal process.
Full reconstruction	The complete payload and related content under explicit legal, regulatory, or governance authority.

A zero-access verifier can confirm that the provenance commitment belongs to the history without reading the payload. A read-capability holder can inspect the details. A public observer can detect later substitution.

PART III

What Can Be Built and Researched

CHAPTER 13

Temporal and Ordering Attacks in Agentic Systems

Frame	Chapter focus
Why	An attacker can cause harm without forging content by exploiting replay, stale state, event reordering, branch equivocation, revocation delay, or witness latency.
How	Object sequence, predecessor links, authority epochs, freshness policy, and independent receipts are evaluated together before an agent acts.
What	The system can distinguish authentic but obsolete information from the latest acceptable state and can expose provisional operation explicitly.

The phrase *timing attack* already has a precise meaning in cryptography: learning secrets from variations in execution time. The attacks considered here are different. They are **temporal attacks on digital history** - attacks that exploit ordering, freshness, race windows, replay, and delayed evidence.

AI agents make these attacks more dangerous because they can observe and manipulate workflows at the speed at which the workflows operate.

A temporal attacker does not always need to change content. It may only need to make the wrong content arrive first, make a valid revocation arrive late, make an old state appear current, or cause different parties to act on different branches.

The main forms are easier to compare as attacks on the relationship between action and evidence.

Attack pattern	How it exploits time or order
Race substitution	Two competing versions appear almost simultaneously, and the attacker makes the malicious branch operational before the legitimate branch is durably witnessed.

Attack pattern	How it exploits time or order
Time-of-check to time-of-use	A capability, price, inventory level, clinical state, or approval changes after it is checked but before the action executes.
Replay	A previously valid message, authorization, or tool result is reused after its intended context or freshness window has ended.
Stale-state injection	A resolver receives an old but internally valid prefix of the history and mistakes authenticity for currency.
Reordering	Individually valid events are presented in a sequence that changes their meaning, such as approval before rather than after modification.
Fork and equivocation	A controller creates two successors to the same state and presents a different branch to each relying party.
Truncation	An intermediary suppresses recent events and returns only an earlier chain that still verifies cryptographically.
Preemption	The attacker anticipates a legitimate commitment and registers a conflicting claim first, exploiting the persuasive force of the earliest public timestamp.
Evidence flooding	A large volume of valid-looking events raises verification cost and hides the transition that matters.
Clock manipulation	Inconsistent clocks, ambiguous time zones, delayed timestamp services, or confused time semantics are used to create a false narrative of order.

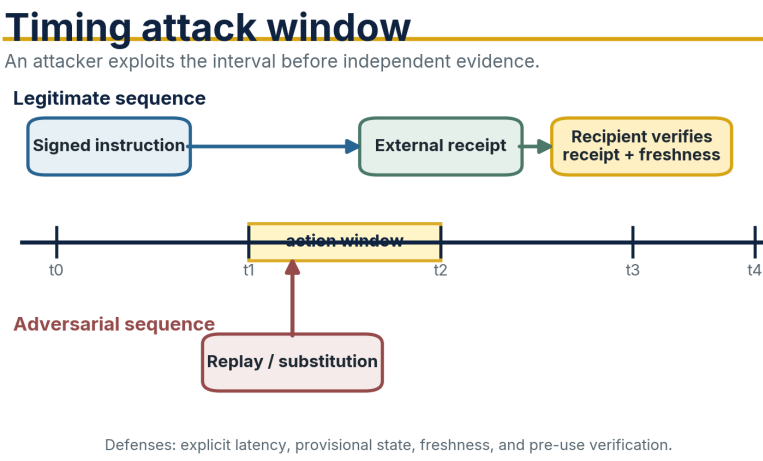

Figure 9. A temporal attack exploits the interval between accepted state and durable independent evidence.

These attacks reveal why timestamps alone are weak. A timestamp is a claim about time. A sequence number is a claim about order. A hash link is a claim about continuity. A witness receipt is evidence that an external service observed the commitment. Strong temporal integrity combines them.

The verifiable event chain should expose the temporal meanings separately.

Temporal evidence	What it establishes
Monotonic sequence	The event's position inside the object's own history.
Previous-event hash	The branch and causal predecessor that the event claims to extend.
Actor time claim	The local assertion about when the action occurred; useful, but not independently conclusive.
External receipt	Evidence that a witness observed the commitment by a registration or checkpoint time.
Authority epoch	The key set, delegation state, and revocation context under which the action was authorized.
Freshness policy	The maximum age or witness maturity acceptable for the intended consequence.

A receiving agent must verify more than signature validity. It must verify that the event extends the expected branch and that no newer witnessed state invalidates the operation.

The first anchor is not the truth. A common mistake is to assume that the first recorded claim wins. Temporal evidence is not epistemic authority. An attacker may preempt a legitimate publisher, and two honest parties may produce conflicting observations.

The correct system evaluates the first claim through the surrounding evidence.

Decision question	Why the first timestamp is insufficient
Actor authority	The earliest claim may have been made by an actor without the mandate to define the relevant fact.
Branch continuity	A claim that does not extend the accepted prior state may be preemptive rather than authoritative.
Witness policy	Different domains require different witnesses, maturity, and consistency guarantees.
Supporting provenance	The observation must be connected to instruments, sources, transformations, or other evidence.
Contradictory claims	Independent observations may reveal uncertainty that no single timestamp resolves.
Dispute rule	Governance decides whether branches are merged, superseded, rejected, or retained as competing claims.

Anchoring helps establish *when* a claim entered the history. It does not decide *which* claim the domain should accept.

Consider a three-second timing attack against an autonomous procurement system. At 10:00:00, a supplier agent publishes a signed quotation for a scarce component. At 10:00:01, an attacker who has compromised a message broker replays an older quotation with a lower price and a broad fulfillment promise. At

10:00:02, the buyer agent accepts the replay and issues a purchase commitment. At 10:00:03, the legitimate quote receives its consortium receipt. Both quotes carry valid supplier signatures because the older one was once genuine.

A content filter cannot solve the problem. The replay is not fake. The critical checks bind the buyer's action to the current branch.

Procurement check	What must be verified
Current branch	Which quotation extends the latest supplier state accepted by the relevant witnesses?
Freshness	Which quotation remains inside the buyer's policy window for price and fulfillment commitments?
Supersession	Was the older quotation revoked, replaced, or made unusable by a new authority epoch?
Input binding	Did the buyer's acceptance commit to the exact hash it evaluated?
Action binding	Did the purchase event commit to both the quotation and the authority state under which it was accepted?

An event-chain protocol can make the attack fail. The buyer refuses any quote that does not extend the latest witnessed sequence. The acceptance event includes the quote hash and a fresh receipt. The purchase commitment becomes inseparable from the exact state used.

Evidence latency as an attack surface. The interval between action and durable commitment should be designed explicitly. Some systems can commit before acting. Others must act first and anchor immediately after. A sensor may buffer events during disconnection. A medical device may prioritize safety over network availability. The system should record the mode and expose the resulting uncertainty.

A disconnected event can remain valid, but it should not pretend to have been externally witnessed at the time of creation. Later anchoring proves that the event existed no later than the registration time; it does not prove the exact claimed creation time.

Explicit representation of temporal evidence is essential because a verifier must distinguish current validity, claimed event time, receipt time, and policy finality.

Multiple clocks, one order. Distributed systems cannot rely on perfectly synchronized clocks. The most robust design treats wall-clock time and causal order as separate dimensions. Hash links and sequence numbers establish local causal order. Independent receipts establish external observation. Secure time sources can add evidence, but no single timestamp should silently carry all meanings.

CHAPTER 14

Proof-Carrying Content Transfer

Frame	Chapter focus
Why	Conventional transfer protocols move payloads while leaving version, authority, provenance, and receipt evidence inside separate platform logs.
How	The transfer binds a content commitment to the sender nano-ledger, capability, witness state, provenance requirements, recipient verification, and acknowledgment.
What	Content may be delivered directly or reconstructed remotely, but the receiving system can verify the exact state and authority before use.

The internet has mature protocols for moving bytes and immature conventions for moving responsibility.

Email, web downloads, APIs, message queues, and object storage can deliver content reliably, but they often leave the receiver to reconstruct context from separate systems. The sender's identity, the exact transmitted version, the applicable authorization, the transformation history, and the receiver's acknowledgment may all exist in different logs.

The early Self-Validating Data research described several application classes that remain relevant because each one requires more than payload delivery. Their enduring contribution is not a prediction that one product will replace email, publishing platforms, or scientific repositories. It is the recognition that communication, delegation, review, and publication need persistent evidence relationships.

Application class	Persistent evidence requirement
Structured messaging beyond ordinary email	A message should bind sender authority, recipient scope, expected response, version, attachments, and acknowledgments so that repeated enterprise interactions become explicit protocols rather than unstructured correspondence.
Personal and autonomous assistants	An assistant acting for a person or organization needs narrow delegated capability, purpose, consent, calendar or resource context, action limits, and a receipt that distinguishes recommendation from executed change.
Executable inter-organizational choreographies	Independent participants need signed state transitions, local validation, aggregate commitments, failure rules, and evidence of which participant performed each contractual step.
Plural-perspective publishing	Contributions, alternative formulations, reviews, endorsements, and editorial selections should remain attributable and recoverable without forcing every participant into one irreversible consensus view.
Scientific publishing and peer review	Data, methods, review comments, revisions, rejected alternatives, and credit should form a verifiable research history while allowing later correction and competing interpretation.
Credential-aware public forums	Pseudonymous contributors may prove relevant qualifications or group membership without disclosing full identity, while moderation, ranking, and sanctions remain inspectable and contestable.

These examples also clarify why a content-transfer layer must support both private detail and shared evidence. A scientific review may be confidential during evaluation but later disclose selected records. An assistant may prove that an authorized task was completed without exposing the user's complete schedule. A public forum may verify a credential without publishing the participant's legal identity.

An OpenDSU-inspired content-transfer system would make the verifiable event chain part of the transfer itself.

The protocol can remain structurally simple while still preserving the evidence required for cross-boundary verification.

The transfer follows a small set of explicit phases.

Transfer phase	Required operation
Define the accountable unit	The sender selects or creates a bounded content unit with a stable identifier and current state commitment. The unit may be a document, dataset, message bundle, model artifact, medical record, contract state, or workflow result.
Build the protected content layer	Content and detailed provenance remain encrypted; large content is divided into content-addressed pieces; and a protected map defines structure and reconstruction keys. The content may remain local, be replicated, or be delivered directly.
Extend the event chain	A transfer event is appended to the nano-ledger and commits to the exact state, prior event, sender, authority, recipient scope, policy, provenance, and freshness conditions relevant to the hand-off.
Obtain witness evidence	The event remains local or is registered in organizational, consortium, transparency, timestamp, or public infrastructure according to the required assurance level.
Deliver constrained authority	The recipient receives only the capability required to reconstruct, append, execute, delegate, or verify the permitted view.
Verify before use	The receiver checks continuity, receipts, capability, authority epoch, freshness, provenance requirements, and domain policy before reconstructing protected content or permitting an external effect.

Transfer phase	Required operation
Return a receipt	The receiver commits to the state received and, where relevant, the state accepted. Later transformations extend the lineage or create a derived unit with an explicit provenance relationship.

Transfer field	What it binds
Unit identity	The accountable object and trust domain to which the transfer belongs.
Exact state	The content or protected-map commitment that the sender intended to make available.
Prior event	The branch and sequence from which the transfer extends.
Sender and authority	The operational actor, controlling authority, capability, and authority epoch.
Recipient scope	The intended recipient, recipient class, or policy-bound audience when disclosure permits it.
Policy and provenance	The requirements that govern use, transformation, receipt, and future evidence.
Freshness or expiry	The period and witness state in which the transfer may still be acted upon.

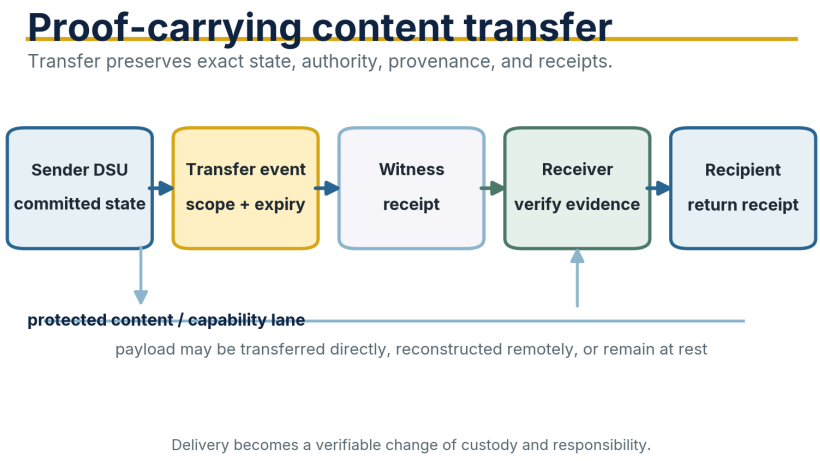

Figure 10. Proof-carrying content transfer preserves committed state, authority, witness maturity, capability, and recipient receipt.

The protocol changes the meaning of delivery from byte transfer to a verifiable change of custody and responsibility.

In an ordinary file transfer, the sender can prove that some bytes were sent only through platform logs. In an event-chain transfer, both parties can prove which version was committed, under which authority, and which receipt followed. The evidence remains portable if the original messaging provider disappears.

The minimum transferable evidence envelope contains five logical elements.

Envelope element	Detailed meaning
Identity	The unit and the verifiable trust domain through which it is resolved.
Commitment	The current state hash, prior-event relation, and policy or authority epoch relevant to the transfer.
Capability	The recipient's authorized relationship to the unit: read, append, execute, delegate, or verify.
Provenance	The required profile and commitment to the detailed encrypted lineage.
Receipt	Evidence from a witness and, after delivery, from the recipient or next custodian.

The content itself may travel inside the envelope, alongside it, or remain at rest while the recipient receives a capability and references.

This flexibility is essential. A terabyte dataset should not be copied merely to prove transfer. A clinical record may remain in the hospital while the patient receives a capability. A public media asset may carry its signed manifest directly. A confidential contract may be exchanged as encrypted bricks with a small public receipt.

Several current standards solve parts of this problem and should be treated as complementary components rather than displaced vocabularies.

C2PA Content Credentials attach cryptographically bound provenance claims to media assets. The IETF SCITT architecture registers signed statements in append-only transparency services and returns receipts. W3C PROV offers interoperable semantics for entities, activities, and agents. Verifiable Credentials express signed claims in an issuer-holder-verifier ecosystem.

The OpenDSU contribution is complementary: the content can remain encrypted and reconstructable as a controlled unit, while identifiers, capabilities, histories, and validation context travel together.

A future system should not compete unnecessarily with these standards. It can use C2PA manifests as provenance payloads for media, SCITT receipts as external anchor evidence, W3C PROV for semantic lineage, and Verifiable Credentials for claims about actors or permissions.

The goal is not one format. It is an interoperable verifiable event chain.

A real transfer protocol must also define failure and fallback behavior when services are unavailable.

If the public witness is unreachable, can a local signed event proceed under a degraded mode? If the receiver cannot reconstruct the complete protected content, can it retain the envelope and request missing bricks later? If the sender's key is revoked after transmission, which policy determines whether the prior event remains valid? If the recipient refuses to issue a receipt, is the send event enough to establish delivery?

These questions are domain-specific. The philosophy requires only that the mode and uncertainty be recorded. A degraded transfer should not masquerade as a fully witnessed transfer.

The requirement for content to retain verifiable context does not mean that every payload must display a complicated badge. Most verification should be invisible in routine operation. The system can show a concise trust summary and allow deeper inspection when needed.

The important change is structural: content should no longer be separated from the minimum evidence needed to use it responsibly.

CHAPTER 15

Digital Trust Ecosystems and Verifiable Governance

Frame	Chapter focus
Why	Cryptography cannot decide who should be trusted, which policies are legitimate, or how affected parties obtain correction and remedy.
How	Digital Trust Ecosystems name roots, witnesses, dependencies, governance versions, dispute rules, and migration procedures while preserving confidential content.
What	The result replaces opaque total trust with explicit technical evidence and contestable institutional responsibility.

No technical system is trustless. What can change is the form in which trust is demanded.

A centralized platform asks users to trust one administrator's database and policy. A public blockchain asks users to trust a consensus protocol, software implementation, economic incentives, and governance community. A self-sovereign system asks users to trust wallets, cryptographic libraries, recovery procedures, witnesses, and domain rules.

The promise of a Digital Trust Ecosystem is not the absence of trust. It is a more explicit and distributed trust contract among independent participants.

An ecosystem may include individuals, companies, regulators, laboratories, devices, public agencies, auditors, and AI agents. They share some data and processes while preserving separate interests. The system must allow cooperation without assuming that any one participant is benevolent or omniscient.

OpenDSU's architecture divides the ecosystem into layers that can be governed separately.

Governance layer	Core questions
Evidence	Which events must be committed? Which witnesses and consistency proofs are acceptable? How long must receipts remain available, and how are conflicting branches handled?
Identity and capability	Which roots bind keys to accountable actors? How are agent identities registered, capabilities scoped, delegations recorded, and compromised authority recovered?
Semantics	Which provenance profiles and validation rules make a clinical, scientific, regulatory, industrial, or financial transition acceptable?
Privacy	Which content and metadata remain encrypted? Under what conditions may auditors reconstruct details, and how are retention and crypto-erasure applied?
Infrastructure	Who operates stores, gateways, registries, domains, and witnesses? What redundancy and migration paths are required?
Dispute and remedy	Who can contest an event, what evidence is admissible, how are false claims superseded, and what remedies follow misuse of a valid key?

These layers should not be collapsed into one administrator. A storage provider should not automatically control identity. A witness should not need to see plaintext. A validator should not necessarily hold decryption keys. An auditor should not be able to change the history it reviews.

This separation of duties is the institutional form of cryptographic least authority.

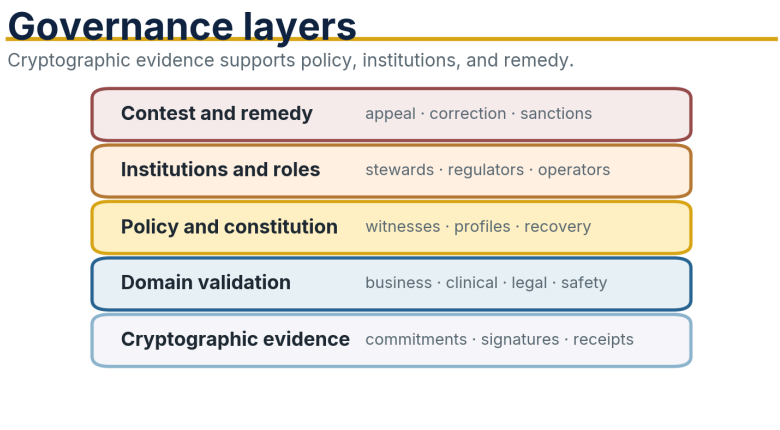

Figure 11. Cryptographic evidence supports but does not replace domain policy, institutions, and contestability.

The word *trustless* is attractive but usually inaccurate; the more precise objective is accountable trust. Human systems cannot eliminate trust because they must interpret meaning, choose rules, and respond to exceptions. A better aspiration is **accountable trust**.

Accountable trust can be tested through four properties.

Property	What it requires
Named dependencies	The system states which identities, validators, witnesses, stores, clocks, and policies it relies upon.
Verifiable claims	Important assertions resolve to deterministic evidence rather than reputation or explanation alone.
Limited authority	Each actor, service, and agent receives only the capability required for its role.
Contestable outcome	Affected parties can inspect evidence, challenge interpretation, obtain correction, and seek remedy.

A system that records immutable events but offers no appeal may be cryptographically strong and politically illegitimate. A system that protects privacy

but prevents lawful audit may be socially unacceptable. A system that reveals every event publicly may destroy the confidentiality needed for cooperation.

Governance determines how these properties are balanced for a particular domain and consequence.

High-consequence ecosystems should avoid dependence on one witness and should use plural custody of consequential evidence. Multiple independent transparency services, periodic cross-anchoring, or threshold receipts can reduce the risk of collusion and failure. The witnesses need not all be blockchains. They may be public authorities, professional bodies, consortium nodes, archival services, or specialized transparency logs.

The important property is that no one actor who benefits from rewriting can silently control all copies of the evidence.

Similarly, encrypted content may be replicated across independent stores or protected through multi-party key recovery. The goal is resilience without universal disclosure.

Regulators often receive evidence too late and in forms created by the regulated party. An event-chain architecture can support continuous or selective audit without granting the regulator permanent plaintext access.

A regulator may verify that required events and provenance profiles exist, that sequence continuity is intact, and that independent receipts were obtained. Detailed content can be disclosed only for selected investigations. This reduces both blind trust and unnecessary surveillance.

AI systems will make decisions that no single human can reproduce manually, making contestability an ecosystem requirement. Contestability therefore cannot depend on obtaining a complete explanation of the model's internal reasoning. It should begin with the external evidence.

Contestability evidence	What an affected party should be able to establish
Model and version	Which model, provider, artifact, configuration, or declared version participated in the decision?
Input commitments	Which data versions, memories, prompts, retrieval sources, or policies were read?
Authority	Which capability, purpose, approval, and authority epoch permitted the operation?

Contestability evidence	What an affected party should be able to establish
Tool activity	Which external tools or services were invoked, and which receipts or outputs were returned?
Committed output	Which exact result became operational, and to which prior state was it linked?
Deployment accountability	Which human or organization authorized the model, workflow, or policy under which the decision ran?
Appeal and correction	Which procedure can suspend, contest, supersede, or remedy the result?

The verifiable event chain provides the procedural record of contestability even when the model remains complex.

Governance must be versioned because a Digital Trust Ecosystem is not finished when the first ledger is deployed. Cryptographic suites age, laws change, attacks evolve, organizations join and leave, and interpretations diverge. Governance events should therefore be versioned and anchored like data events.

The domain constitution - root keys, witness sets, required profiles, recovery procedures, dispute rules, and migration policies - should have its own verifiable history.

This is where OpenDSU's openness by design becomes essential. The core should standardize the minimum evidence relationships while allowing domains to evolve their own semantics and infrastructure.

CHAPTER 16

LightDSU as a Minimal Reference Architecture

Frame	Chapter focus
Why	The historical OpenDSU stack can obscure the minimum relationships that an implementation must preserve and can raise the cost of experimentation.
How	LightDSU implements encrypted bricks, a protected BrickMap, lkey/rkey/lza capabilities, signed EventSSI history, access control, policy, key epochs, and encrypted provenance locally.
What	The reference demonstrates local-first evidence and optional assurance escalation without claiming that a local file provides independent consensus.

Implementations often accumulate adapters, deployment patterns, compatibility layers, operational services, and historical decisions beyond the minimum required by their conceptual model. Some of that complexity is justified, but it can obscure the invariants that a new implementation must preserve. LightDSU is valuable because it asks a precise architectural question: what is the minimum implementation complexity required to preserve the essential OpenDSU relationships? Its proposed core is intentionally limited.

Local core	What it preserves
Encrypted bricks	Content-addressed encrypted pieces whose integrity can be checked independently of the storage path.
Encrypted BrickMap	A full-snapshot protected structure that reconstructs the current committed version.
Capability family	Compact control, read, and verification-only relationships expressed as lkey, rkey, and lza.

Local core	What it preserves
Append-only anchor file	One signed EventSSI per line, linked to the previous event and inspectable with ordinary tools.
Semantic event set	Explicit versions, grants, revocations, access logs, provenance, policy updates, and key epochs.
Encrypted provenance payloads	Rich profile-based lineage remains private while its commitment enters the event chain.

No distributed ledger is required in the local core. No blockchain node is required. No large server platform is required. The initial history can be represented by a local append-only file.

This is not a retreat from OpenDSU. It is a clarification of it.

The earlier architecture sometimes allowed the implementation substrate to dominate the explanation. LightDSU makes the relationship between object and history visible. The data unit is reconstructed from encrypted pieces. The event chain records how its state and authority change. Stronger external anchoring can be added later.

The conceptual classification depends on the admitted event semantics. Version commitments and control transitions constitute the anchor or nano-ledger. When the same EventSSI envelope also carries domain-specific operations whose replay determines valid state, it can implement a compact micro-ledger. LightDSU therefore demonstrates that one local event format can preserve the distinction without requiring separate infrastructure for each layer.

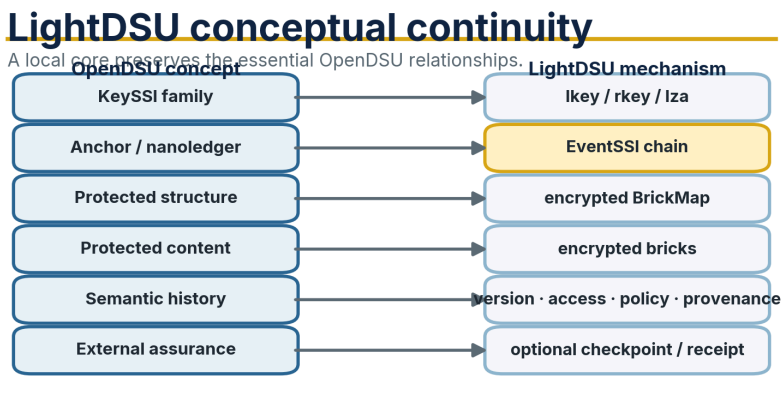

Figure 12. LightDSU preserves the OpenDSU conceptual relationships through a minimal local reference architecture.

The continuity can be expressed through preserved invariants rather than repeated implementation names.

Invariant	How LightDSU preserves it
Bounded unit	The filesystem interface exposes one accountable logical object rather than unrelated local files.
Protected structure	Every committed version references a complete encrypted BrickMap that can reconstruct that version.
Hierarchical capability	Ikey, rkey, and lza retain the separation among control, reading, and zero-access verification.
Append-only history	Each EventSSI links to its predecessor and is signed under the anchor-control key.
Actor accountability	The event can carry an operational actor signature distinct from the signature that controls object continuity.
Private provenance	Detailed profile payloads remain encrypted and are referenced from compact signed events by hash.

This design suggests a general strategy for the future: **local-first evidence, optional assurance escalation**.

A personal application can use only the local log. A small company can seal logs into an organizational transparency service. A regulated consortium can require receipts for cross-boundary events. A public institution can anchor selected checkpoints into an independent public ledger.

The application code does not need to know which level is ultimately used. It commits material events to the same logical event chain.

Lightness is not the absence of history. A lighter OpenDSU family can support several temporal modes without pretending that they provide the same assurance.

Lightweight mode	Appropriate use
Versionless current state	Simple local or edge objects whose only claim is the current value. They avoid history cost, but cannot support historical reconstruction or strong dispute about prior states.
Local EventSSI history	An accountable object whose material changes, access, provenance, policy, and key epochs are recorded on the host. This is a meaningful first evidence tier, not an independent witness.
Externally witnessed history	Selected events or checkpoints receive organizational, consortium, transparency-service, or public receipts when cross-boundary reliance requires stronger assurance.

The distinction prevents two opposite mistakes. History should not be imposed where no evidentiary claim is needed, and a versionless object should not be promoted as though it offered the continuity of a nano-ledger. The chosen mode belongs in the unit's policy and verification report.

The security budget should follow consequence. Heavy architectures frequently force low-risk and high-risk operations through the same infrastructure. This increases cost and encourages developers to bypass the mechanism entirely.

A lighter design can classify consequence instead of routing every event through maximum infrastructure.

Example consequence	Proportionate evidence
Private local note	A signed local history may be enough when no external party relies on it.
Internal approval	An organizational checkpoint separates the decision from the application that requested it.
Cross-institution clinical result	Consortium witnessing and regulated provenance support reliance, review, and dispute between institutions.
Public safety directive	Independent multi-witness registration protects against unilateral withdrawal, equivocation, or institutional capture.

The same philosophy supports all four.

A compact anchor file is not automatically more secure than a distributed ledger, but its simplicity improves auditability. A security reviewer can inspect the event encoding, signature rules, sequence checks, and storage layout. Fewer components mean fewer hidden trust transitions.

This is especially important for cryptographic infrastructure. Complexity becomes security debt when only a small group can explain the system.

A local LightDSU core can still interoperate and should not be interpreted as an isolated system. Event formats can be exported. Checkpoints can be registered in SCITT-like transparency services. Media provenance can include C2PA claims. Actor identities can use DIDs or organizational certificates. Provenance payloads can align with W3C PROV, FHIR, RO-Crate, GxP, or AI experiment profiles.

The local library becomes a producer and consumer of portable evidence rather than a miniature closed world.

A lightweight implementation must not hide the assurance it omits. A local append-only file does not protect against a fully compromised host. One control key does not provide institutional recovery. A complete BrickMap snapshot may leak access patterns through update size. A single clock cannot establish independent time. A local verifier cannot detect a fork shown only to another party.

The correct claim is narrower: the local core establishes a clean evidence object that can be strengthened by additional witnesses, hardware, policies, and replication.

Technical point	Explanation
Preserve the invariants across implementations	OpenDSU should not demand that every adopter install the historical stack. It should demand that every implementation preserve the essential relationships among content, capability, history, provenance, and reconstruction.

CHAPTER 17

OpenDSU Beyond the Enterprise Blockchain Cycle

Frame	Chapter focus
Why	The decline of enterprise-blockchain promotion can cause useful mechanisms to be discarded together with inflated claims and unsuitable deployments.
How	The chapter separates durable requirements for integrity, attribution, ordering, confidentiality, and portability from tokens, global consensus, and technology branding.
What	OpenDSU remains relevant as an implementation-neutral framework that can use transparency services, organizational ledgers, local logs, or future witnesses.

Technology fashion is a poor measure of architectural necessity. Enterprise blockchain passed through a familiar cycle. It was first presented as a universal transformation. Every consortium needed a chain; every database problem was renamed a trust problem; every proof of concept promised disintermediation. Many projects discovered that governance, integration, incentives, privacy, and operational ownership were harder than selecting a ledger.

By 2026, attention and budgets have moved decisively toward generative and agentic AI. Enterprise blockchain remains in selective domains, but it no longer occupies the center of the conversation. The change can be described without nostalgia: the fashion receded.

The underlying need did not disappear; the durable problems have returned under different names.

Current vocabulary	Underlying need
Signed metadata and content credentials	Bind assertions about origin and transformation to exact content.

Current vocabulary	Underlying need
Transparency logs and audit trails	Constrain silent rewriting and provide independently checkable sequence evidence.
Software supply-chain attestations	Connect builds, dependencies, environments, and releases to accountable producers.
Verifiable credentials and agent identities	Express signed claims about actors while keeping identity distinct from operational authority.
Execution receipts	Record what a tool, model, service, or agent actually committed to doing or returning.
Model and dataset provenance	Preserve the lineage needed to evaluate, reproduce, or contest AI outputs.
Policy-as-code	Make the rules governing valid transition executable and versioned.
Confidential-computing evidence	Show which protected environment executed an operation without publishing its private inputs.

These are anchoring problems even when no one uses the word blockchain.

The lesson is not that blockchain was secretly right about everything. It was wrong whenever it confused global consensus with trust, tokens with governance, immutability with truth, or decentralization with legitimacy. The lesson is that the technology exposed a real deficiency in conventional systems: mutable institutional memory is often controlled by the same party whose behavior the memory is meant to audit.

A ledger is one method of introducing an external constraint.

OpenDSU's most durable choice was therefore blockchain agnosticism. The purpose of the ledger was primarily notarization and ordering, not universal computation. The confidential content and validation remained off-chain. This separation allows the anchoring substrate to change with the era.

A future OpenDSU implementation can choose among several witness substrates.

Witness substrate	What it contributes
Local hash-chained event file	Immediate, inspectable evidence with minimal operational complexity and no independent anti-rewriting guarantee.
Append-only database proof	Familiar operations with stronger internal tamper evidence and efficient querying.
Merkle transparency log	Scalable inclusion and consistency proofs for many signed statements.
SCITT service	Standardized registration and receipts for signed statements across supply chains and organizations.
Consortium ledger	Shared governance and ordering among a defined set of participants.
Public blockchain	Broad observability and independence when cost, privacy leakage, and public permanence are acceptable.
Secure timestamp authority	External evidence that a commitment existed no later than a registered time.
Composite witness set	Several substrates combined to avoid one operator, jurisdiction, or technology becoming the sole historical authority.

The philosophical requirement is independent evidence, not brand loyalty.

Mature security mechanisms are commonly integrated as background infrastructure rather than exposed as user-facing concepts. TLS, certificate transparency, signed software packages, hardware roots of trust, and database write-ahead logs illustrate this pattern. Anchored content may follow the same operational path, and future users may never need to see the word ledger. A document will simply show that its source and edit history are verifiable. An AI agent will refuse a stale capability automatically. A regulator will request a cryptographic receipt instead of a screenshot. A scientific paper will link to a reproducible evidence package. A hospital will prove that a result was not silently revised after treatment.

Successful deployment makes these mechanisms routine infrastructure.

The proliferation of autonomous AI changes the cost equation. Enterprise blockchain often struggled because the immediate business benefit of strong shared history did not justify the integration cost. AI changes both sides of the equation.

AI effect	Change in the evidence economy
More machine action	Decisions, synthesized content, and cross-organizational workflows occur at a volume and speed that manual logging cannot reconstruct.
Cheaper deception	Plausible documents, identities, explanations, and tool outputs can be produced and adapted at scale.
More expensive retrospection	One decision may depend on many models, retrieval sources, prompts, tools, and dynamic policies whose state disappears quickly.
Cheaper evidence operations	Agents can help classify material events, populate provenance, summarize deterministic reports, and identify missing commitments, provided the cryptographic core does not depend on the model's story.

The economic argument for anchored information may therefore become stronger precisely when the blockchain label becomes weaker.

The concepts can reappear under different technical labels. The phrase *enterprise blockchain* invited people to debate the substrate before agreeing on the problem. A better vocabulary begins with the desired properties.

Desired property	Practical question
Verifiable history	Can a relying party detect truncation, substitution, and retrospective rewriting?
Accountable state transition	Can the system identify the prior state, actor, authority, policy, and resulting commitment?
Privacy-preserving evidence	Can integrity and lineage be checked without publishing confidential content or unnecessary metadata?

Desired property	Practical question
Portable capability	Can authority remain meaningful outside the original application and be attenuated to the task?
Independent witness	Is at least one copy of consequential evidence outside the control of the actor who benefits from rewriting?
Reconstructable provenance	Can authorized parties recover enough process context to assess, reproduce, or contest the result?

OpenDSU should be presented through these outcomes. Blockchain is one witness technology among several.

A second hype cycle should be avoided because the return of provenance and transparency infrastructure will attract new exaggerations. Content credentials may be presented as a cure for misinformation. Agent receipts may be presented as proof of safe behavior. Transparency logs may be presented as proof of truthful statements. The same conceptual mistakes can return under modern labels.

The OpenDSU research programme should therefore maintain a precise and limited conformance boundary.

Mechanism	Honest limitation
Anchoring	Proves registration, continuity, and sometimes ordering; it does not prove the truth of the underlying claim.
Signature	Proves control or use of a key under a cryptographic convention; it does not by itself establish moral, legal, or biological identity.
Provenance	Improves assessment and accountability; it does not manufacture certainty from weak sources.
Decentralization	Distributes power and failure, but can also distribute responsibility so widely that no remedy remains.

Mechanism	Honest limitation
Tamper-evident history	Preserves errors and abuse as well as correct actions; governance must decide what later states accept.
Auditability	Matters only when evidence remains available and institutions can investigate, contest, correct, and sanction.

CHAPTER 18

Research Programme for AI Security and Verifiable Systems

Frame	Chapter focus
Why	AI-agent security, identity, authorization, provenance, and transparency remain active standardization and research problems rather than solved products.
How	The chapter formulates testable hypotheses, threat models, evaluation dimensions, interoperability requirements, and privacy constraints.
What	A rigorous programme should produce independent implementations, conformance vectors, attack evaluations, and explicit statements of residual trust.

OpenDSU provides a conceptual framework, not a completed security theorem. Its relevance to AI depends on disciplined research that tests whether object-scale histories, capabilities, provenance, and independent witnessing improve real systems under realistic attack and operational constraints.

The external research environment has moved in a compatible direction. NIST launched an AI Agent Standards Initiative in 2026 with explicit work on secure interoperability, agent identity, authorization, and security evaluations. NIST's Cyber AI Profile organizes work around securing AI components, conducting AI-enabled defense, and thwarting AI-enabled attacks. Transparency architectures such as SCITT register signed statements and return receipts. C2PA provides provenance structures for digital media. These efforts solve different problems, but together they show that identity, authorization, provenance, receipts, and evidence portability are active areas of standardization rather than historical blockchain concerns.

The OpenDSU contribution should be framed as a set of research hypotheses.

Research hypothesis	Question to test
Object-scale histories reduce ambiguity	Do per-object nano-ledgers make stale state, truncation, replay, and conflicting branches easier to detect than conventional application logs?
Capability-derived authority limits agent damage	Can task-specific read, append, export, and tool capabilities reduce the blast radius of prompt injection, model compromise, or orchestration error without making systems unusable?
Provenance profiles improve reproducibility	Which minimum commitments to datasets, prompts, tools, models, parameters, environments, and evaluators materially improve investigation and contestability?
Progressive witnessing supports adoption	Can systems begin with local signed histories and add independent receipts without redesigning object identity, storage, and verification?
Self-validating micro-ledgers improve cross-domain workflows	Can independent participants validate the part of an executable choreography they are authorized to see while preserving aggregate process evidence?
Plural trust domains improve resilience	Can BDNS-like resolution support migration and heterogeneous infrastructure without replacing one central dependency with an opaque naming authority?

The first experimental priority is a clear threat model. Research should distinguish compromise of the local host, theft of a control key, malicious but authorized actors, dishonest witnesses, resolver attacks, delayed receipts, cross-agent prompt injection, model substitution, tool-output forgery, and governance capture. A mechanism that resists one category should not be described as a general solution.

The second priority is measurable verification latency. In an autonomous workflow, evidence that arrives after the irreversible action may have forensic value but little preventive value. Experiments should measure the interval from semantic operation to local commit, from local commit to external receipt, and from receipt

to recipient verification. Policies should be evaluated under disconnection, congestion, witness failure, and adversarial scheduling.

The third priority is evidence minimization. Rich provenance can become surveillance. Recording every prompt, tool call, user identifier, and intermediate state may expose personal data, trade secrets, security procedures, or model internals. Research should compare full disclosure, encrypted payloads, selective disclosure, commitment-only records, aggregate checkpoints, and zero-knowledge techniques. The objective is not maximal logging but sufficient evidence for a declared reliance and dispute model.

Evaluation dimension	Required measurement
Security effect	Attack success rate, time to detection, fork or replay detection, recovery behavior, and residual trust assumptions.
Operational cost	Commit latency, receipt latency, storage growth, bandwidth, resolver load, and validator execution cost.
Privacy effect	Metadata leakage, cross-object correlation, disclosure surfaces, and the consequences of long-term commitment retention.
Interoperability	Independent implementations producing compatible commitments, capabilities, provenance payloads, and verification reports.
Usability	Human understanding of assurance states, capability grants, failures, and contest procedures.
Governance	Ability to rotate keys, replace witnesses, migrate domains, correct records, preserve competing claims, and provide remedies.

A reference implementation should not be the only oracle for conformance. The framework needs normative test vectors for event canonicalization, capability derivation, AnchorID and nano-ledger validation, BrickMap reconstruction, SVD replay, provenance validation, witness receipts, and domain migration. At least two independent implementations should process the same objects and produce equivalent verification outcomes.

AI provenance requires particular care. A provider label is not an immutable model artifact. A model-weight hash does not capture a remote safety layer, retrieval service, or tool. A prompt hash does not reveal the prompt and may not capture dynamic system messages. A container commitment does not prove hardware state or external dependencies. Profiles should distinguish what is directly committed, what is provider-asserted, what is unavailable, and what was independently attested.

The same discipline applies to agent identity. A DID, certificate, or public key establishes a verification method, not organizational authority or human intent. Research should model agent roles, delegation paths, purpose, spend or action limits, review requirements, key epochs, and emergency suspension. The system must retain the distinction between the service controlling the object history and the operational agent initiating a particular action.

Cross-agent systems also need compositional security. An upstream agent may provide a valid result under a policy that the downstream domain does not accept. A tool receipt may be genuine but insufficient for the consequence. A choreography should therefore produce locally evaluable evidence at each boundary, not one global assertion that the workflow is trusted.

Open research remains at the intersection with law and institutional governance. Retention of cryptographic commitments may conflict with deletion expectations. Selective disclosure may be incompatible with some audit rights. Automated refusal based on provenance policy can entrench errors or exclusion. Independent witnesses can centralize. A technically verifiable history can support an unjust process. Research must include affected-party contestability and not only system-owner assurance.

The appropriate outcome is a set of falsifiable results: which OpenDSU relationships improve which classes of AI system, at what cost, under which trust assumptions, and with what new risks. The conceptual framework is valuable precisely because it allows those questions to be studied without committing the research programme to one ledger or codebase.

CHAPTER 19

Evaluation Criteria, Limits, and Responsible Adoption

Frame	Chapter focus
Why	A verifiable history can still be false, coercive, privacy-invasive, expensive, or governed by institutions that affected parties cannot challenge.
How	The chapter evaluates residual trust, metadata leakage, key compromise, deletion, proof longevity, cost, interoperability, and governance remedies.
What	A responsible implementation publishes a precise verification profile and makes omitted assurances, failure modes, and contest procedures explicit.

The philosophy developed in this book can be reduced to a set of design principles. They are not an API specification. They are tests that future libraries, wallets, agent platforms, and digital trust ecosystems should be able to pass.

Design principle	Test for a future implementation
Anchor material change by default	Creation, revision, transfer, approval, revocation, policy change, key change, and consequential analysis normally produce compact evidence; materiality policy prevents indiscriminate logging.
Keep confidential content outside public evidence	Witnesses receive commitments and minimum metadata, while detailed content and provenance remain encrypted and selectively disclosed.
Separate existence from truth	Interfaces state exactly what cryptography proves and which claims still require domain judgment, observation, or institutional accountability.

Design principle	Test for a future implementation
Use capabilities, not ambient authority	Agents and applications receive the narrowest right needed: read, append, execute, grant, revoke, or verify.
Begin local and escalate assurance	A local signed history is a valid first tier; independent witnesses and consensus are added according to consequence.
Preserve cryptographic agility	Identifiers and events declare algorithm versions and support key, signature, encoding, and post-quantum migration without breaking history.
Use plural witnesses for high consequence	No actor who benefits from rewriting controls every copy of critical evidence.
Make correction visible and possible	Errors are superseded through explicit corrective events; preserved evidence is not confused with continued acceptance of the claim.
Minimize metadata leakage	Names, timing, size, access patterns, and public anchor frequency are included in the privacy threat model.
Require provenance proportional to consequence	The operation, domain, law, and potential harm determine which profile and evidence are mandatory.
Be machine-verifiable and human-contestable	Deterministic checks run before automation depends on the state, while people receive understandable reports and practical appeal.
Treat implementations as replaceable	Libraries, ledgers, clouds, and runtimes may change while the relationships among content, capability, history, provenance, reconstruction, and governance remain stable.

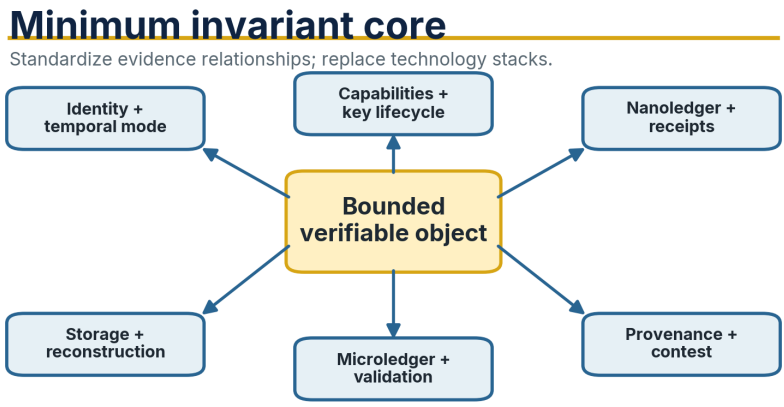

Figure 13. Stable conceptual relations can support replaceable identifier, storage, ledger, runtime, and witness implementations.

These principles suggest a reference pattern for future systems.

At the center is an **Anchored Content Unit**. It may be implemented as a DSU, a LightDSU, or another compatible object. It contains or references encrypted content and a protected structural map.

The unit has a **verifiable event chain** of signed events. Each event commits to the prior event and to the material change. The event can include an actor signature, an anchor-control signature, a policy version, and an optional witness receipt.

Access is mediated by a **capability family**. Strong capabilities can derive weaker ones. The security context performs cryptographic operations without exposing raw keys.

Discovery occurs through a **verifiable trust domain**. The resolver learns which storage, witnesses, algorithms, and policies apply.

Reconstruction occurs inside a **wallet, enclave, or controlled agent environment**. The environment verifies the history, resolves the capability, obtains the necessary bricks, and enforces provenance requirements.

High-level workflows compose units through **mounting and choreography** rather than copying everything into one central database.

External standards should enter the architecture through explicit profiles and receipts rather than be reimplemented under new names.

External standard family	Role in the reference pattern
C2PA	Cryptographically bound provenance claims and manifests for media and other content assets.
SCITT	Transparency-service registration and receipts for signed statements across organizations and supply chains.
W3C PROV	Interoperable semantics for entities, activities, agents, derivation, association, and revision.
Verifiable Credentials	Signed claims about actors, roles, qualifications, status, or delegated relationships.
Sector profiles	Clinical, regulatory, scientific, genomic, research, and industrial requirements expressed as domain-specific provenance and validation rules.

A practical adoption sequence. Organizations should not begin by replacing core systems. A realistic adoption path is incremental.

Adoption stage	Outcome
Choose the evidence gap	Select one high-consequence object whose history, authority, or provenance is currently difficult to defend.
Define materiality	Name the transitions that matter and the minimum provenance, freshness, and retention rules.
Create a local event chain	Produce signed events beside the existing system without requiring an immediate platform replacement.
Bind state and authority	Connect every event to the exact content version, actor, capability, policy, and authority epoch.
Add independent witness	Obtain external receipts for cross-boundary, disputed, or regulated consequences.

Adoption stage	Outcome
Introduce controlled access	Replace broad exports and account sessions with capabilities for the required view or operation.
Move content selectively	Use encrypted reconstructable units where portability, privacy, or independent verification justifies the change.
Exercise failure governance	Test migration, revocation, correction, evidence loss, fork detection, and dispute before scaling.

This sequence treats evidence as an overlay before it becomes infrastructure. It allows value to be demonstrated without a grand replacement program.

The verification report as a public interface. Future systems should standardize a compact verification report. The report should not merely say "valid" or "invalid." It should separate dimensions so that relying parties can make policy decisions without accepting false certainty.

Report dimension	What is disclosed
Content commitment	Whether the reconstructed or presented state matches the expected cryptographic representation.
History continuity	Whether the event extends the accepted branch without missing or inconsistent predecessors.
Witness maturity	The latest receipt, checkpoint, pending state, and assurance policy satisfied.
Actor evidence	The key, identity relationship, device, service, or agent evidence associated with the event.
Capability authorization	The operation, scope, purpose, time, and authority epoch under which the actor was permitted to act.
Provenance completeness	Which required profiles and fields are present, encrypted, disclosed, or missing.

Report dimension	What is disclosed
Domain validation	Which historically bound rules were evaluated and under which execution assumptions.
Retention and decryptability	Whether the content is available, lawfully recoverable, expired, or intentionally crypto-erased.
Conflicts	Any forks, contradictory witnesses, stale state, unresolved dependencies, or disputed claims.
Assurance summary	The witnesses, algorithms, verification depth, and remaining uncertainty on which the result depends.

An AI system can summarize this report, but the report itself should be deterministic and independently reproducible.

A project that implements only a small core can still participate in the OpenDSU conceptual framework if its conformance claim is explicit.

Minimum element	Why it is essential
Bounded encrypted content unit	Gives accountable meaning and privacy a stable object on which to operate.
Signed append-only history	Links material events to exact states and makes retrospective rewriting detectable.
Capability separation	Distinguishes control, reconstruction, bounded operation, and verification without disclosure.

Additional functions should remain outside, or be layered above, that minimum interoperable core.

The framework should be evaluated through its limits as well as its intended properties.

Limit or trade-off	Architectural consequence
History does not establish truth	A coherent signed sequence can contain false measurements, biased models, fabricated claims, or unlawful decisions. Domain evidence and institutional review remain necessary.
Signatures do not establish intent	Malware, coercion, automation error, or key compromise can produce valid signatures. Security Contexts, confirmation policy, threshold authority, and recovery are required.
Anchoring does not prevent all forks	A dishonest controller can create branches before independent witnessing or show different local histories. Receipt policy, multi-witness checking, and conflict procedures are needed.
Metadata can violate privacy	Event type, timing, frequency, identifiers, and relationships can reveal sensitive behavior even when payloads are encrypted. Batching, scoped hashes, private witnesses, and minimization must be considered.
Capability sharing complicates revocation	A copied read capability may remain effective after disclosure unless content is re-encrypted, access is mediated, or key epochs change. Offline sovereignty and immediate revocation are in tension.
Long-lived proof requires maintenance	Algorithms, formats, resolvers, witnesses, and validator dependencies decay. Migration proofs, archival software, and succession governance are part of evidence preservation.
Deletion and audit can conflict	Retaining commitments supports accountability but may conflict with legal or ethical deletion expectations. Domains must define retention, crypto-erasure, aggregation, and lawful access explicitly.

Limit or trade-off	Architectural consequence
Verification has a cost	High-frequency histories, rich provenance, and external receipts consume storage, computation, network capacity, and human attention. Materiality and assurance must be proportional to consequence.
Evidence can become coercive infrastructure	A mandatory universal evidence regime can exclude small actors, enable surveillance, or make contest impossible. Open formats, plural witnesses, proportionality, and remedy are essential.

A technically credible implementation should therefore publish a verification profile rather than rely on a general claim that it is OpenDSU-based. The profile should identify the object boundary, temporal mode, nano-ledger format, micro-ledger or SVD semantics if present, capability model, storage and reconstruction assumptions, witness policy, provenance requirements, key lifecycle, privacy leakage, failure modes, and contest procedures.

The minimum conceptual conformance claim is also limited. A system may be inspired by OpenDSU while omitting independent witnessing, programmable validation, or portable applications. That can be acceptable if the omission is explicit. The danger begins when a current-state encrypted container is described as though it provided a versioned nano-ledger, or when a signed log is presented as though it provided independent freshness and truth.

OpenDSU should remain open by design in a technical rather than rhetorical sense. The stable boundary should be small enough for multiple implementations: object identification, protected content commitments, capability semantics, nano-ledger continuity, optional micro-ledger validation, reconstruction, provenance binding, and witness receipts. Domain communities should remain free to define data models, professional rules, legal requirements, and assurance levels.

The book's final position is therefore conditional. OpenDSU concepts can become structural components of future digital systems if they remain interoperable, minimally sufficient, privacy-preserving, and contestable. They will fail if they become another proprietary platform, another claim that one ledger creates truth, or another compliance system that records everything while giving affected parties no effective remedy.

EPILOGUE

Verifiable History as Digital Infrastructure

The expansion of autonomous systems changes the role of digital records. Information is no longer only read by people and stored for later review. It becomes an immediate input to machines that select tools, authorize transfers, update institutional state, and communicate conclusions to other machines. The quality of the surrounding evidence therefore becomes part of operational safety.

OpenDSU offers a coherent way to organize that evidence. A Data Sharing Unit provides a bounded logical object. Encrypted bricks and a protected BrickMap separate custody from interpretation. KeySSI families express control, reading, and verification authority. The anchor is the nano-ledger that preserves compact object continuity. A micro-ledger can provide richer domain operations, and Self-Validating Data is one implementation pattern for binding such a history to replay and validation rules. BDNS resolves plural infrastructure and policy. Security Contexts constrain where authority is exercised. Provenance profiles connect transformations to domain expectations.

No single mechanism is sufficient. A hash does not prove truth. A signature does not prove intent. A witness does not prove scientific validity. A model trace does not establish fairness. The value lies in composition: each mechanism answers a limited question, and the verification report makes the remaining uncertainty visible.

The framework is deliberately compatible with technological change. A nano-ledger may be stored in a local file, a transparency service, a consortium ledger, or another future witness. A micro-ledger may use a chain, DAG, CRDT, event store, or verifiable computation method. A DSU may be reconstructed in a browser, mobile wallet, enterprise agent, trusted execution environment, or another controlled runtime. The implementation can evolve while the evidence relationships remain stable.

The practical objective is not to notarize everything. It is to make materiality, authority, continuity, provenance, and assurance explicit enough that high-consequence systems can verify before acting and can reconstruct the basis of

an action after a dispute. This is particularly important when AI systems operate faster than institutional review.

Verifiable history should therefore be treated as digital infrastructure: compact enough to be routine, private enough to be proportionate, portable enough to outlive a platform, and governed well enough to support correction and contest. OpenDSU's essential philosophy is a proposal for that infrastructure.

APPENDIX A

Essential Vocabulary

Term	Meaning
Anchor / nano-ledger	The compact append-only ledger of one DSU or bounded digital object. It is identified by an AnchorID and records signed version commitments and permitted control transitions.
AnchorID	The stable identifier used to locate and validate the nano-ledger, normally related to the public control material admitted by the anchor rules.
Anchoring	The operation of extending a nano-ledger or registering a nano-ledger commitment with an organizational, consortium, transparency, timestamp, or public witness.
Brick	An encrypted content-addressed unit whose integrity can be checked from its cryptographic identifier independently of its storage path.
BrickMap	The protected structure that maps the logical contents of a DSU to bricks, keys, metadata, and reconstruction relationships for a committed version.
BDNS	Blockchain Domain Name System; the OpenDSU concept for resolving ledger or trust domains to storage, anchoring, validation, and other infrastructure configuration.
Capability	Cryptographically represented authority to identify, read, modify, delegate, execute, export, or verify an object under specified constraints.

Term	Meaning
Data Sharing Unit (DSU)	A bounded reconstructable digital object that can relate protected content, structure, identity, capability, provenance, policy, validation, and verifiable continuity.
EventSSI	The compact signed event representation proposed by LightDSU for versions, access changes, provenance, policy, key epochs, and related object history.
Far-chain	Operational or analytical data stores that are only indirectly related to an anchored object, such as search indexes, vector databases, dashboards, and enterprise data warehouses.
KeySSI	Key Self-Sovereign Identifier; an identifier and capability family used to locate, decrypt, reconstruct, verify, or control a DSU.
micro-ledger	A scoped programmable ledger for one data structure, process, relationship, or smart-contract-like object. It can admit domain-specific operation types and validation rules.
micro-ledger SVD	A Self-Validating Data implementation whose command history and replay rules form a micro-ledger, normally stored off-chain and committed through a DSU anchor or another witness.
Near-chain	Protected data stored outside a shared ledger but deliberately and verifiably bound to it through nano-ledger commitments or checkpoints.
Notarization by default	The principle that a material state transition normally creates canonical signed evidence and obtains the witness maturity required by policy.
Proof-carrying content	Content delivered with, or resolvable to, machine-verifiable evidence about exact state, authority, history, provenance, policy, and witness receipts.

Term	Meaning
Provenance profile	A declared schema and validation policy for domain-specific lineage, access, transformation, approval, governance, or AI execution metadata.
Reconstruction	The authorized process of resolving capability and domain configuration, verifying the nano-ledger, obtaining and decrypting the BrickMap and bricks, validating the object, and materializing the permitted view.
Security Context	The controlled environment in which identities, keys, capabilities, sensitive records, policy, and cryptographic operations are interpreted.
Self-Sovereign Application (SSApp)	Portable application code and data executed in a user- or organization-controlled Security Context instead of permanently owning the data it interprets.
Self-Validating Data (SVD)	A possible micro-ledger implementation that binds a data structure to its command history, provenance, replay or state-transition rules, and validation artifacts so that a compatible environment can derive and evaluate state locally.
Secret smart contract	A placement pattern in which confidential validation logic executes off-chain near protected data while compact state commitments and ordering evidence are externally witnessed.
Versionless unit	A bounded object that exposes current state without claiming append-only historical continuity or reconstructability of prior versions.
Verification-only capability	Authority to inspect nano-ledger continuity or existence without decrypting the protected content.

Term	Meaning
Witness receipt	Cryptographic evidence that an external service or ledger registered a commitment according to declared ordering and consistency rules.

SOURCES

Sources and Further Reading

The sources below identify the principal conceptual lineage and the current research context. The book interprets them comparatively; inclusion does not imply endorsement by an external standards body.

Source	Relevance to this book
OpenDSU public documentation and RFC index, OpenDSU.org	Current public map of DSU, KeySSI, Brick Storage, anchoring, mounting, reconstruction, BDNS, Security Contexts, enclaves, SSApps, VersionLessDSUs, Optimistic Blockchain Anchoring, SVD, and draft directions.
Lenuta Alboaie and Sinica Alboaie, editors, *OpenDSU Yellow Book: Concepts and Design*, first edition, 2023	Primary technical and conceptual lineage for Digital Sovereignty, near-chain storage, DSUs, KeySSIs, anchors, micro-ledgers, BDNS, wallets, enclaves, and Self-Sovereign Applications.
OpenDSU Anchoring, RFC-005	Defines blockchains primarily as notarization mechanisms for DSUs, describes AnchorIDs and light explicit anchors, and separates compact on-ledger commitments from off-chain history and validation.
OpenDSU Self-Validating Data, RFC-036, public review document	Defines SVD properties, history representations, primitive and state operations, micro-ledger SVD, executable choreographies, command replay, and the SVD lifecycle.
Cosmin Sînică, *Self Validating Data*, research article	Research formulation of SVD as a conceptual model beyond one implementation, including provenance, integrity, self-validation, composition, replay, and collaborative application directions.
Sînică Alboaie, Axiologic Research, *OpenDSU & Tokenisation*, version 1.0, December 2023	Clarifies that an anchor is a nano-ledger, explains its append semantics and freshness role, distinguishes nano-ledgers from micro-ledgers represented by SVDs, and describes stable identity under control transition.
***LightDSU v1 - Self-contained Technical Specification*, design note**	Provides a local-first reference with lkey, rkey, lza, encrypted BrickMap, encrypted bricks, EventSSI history, access, provenance, policy, key epochs, and actor versus anchor signatures.

Source	Relevance to this book
***LightDSU v1 Extension - Provenance Profiles and Regulatory Metadata*, design note**	Defines compact provenance events and encrypted profile payloads for W3C PROV, FHIR, GxP, RO-Crate, genomic data use, ISO 8000, OECD GLP, and AI/ML experiments.
NIST, AI Agent Standards Initiative, 2026	Establishes current work on secure and interoperable AI agents, including agent security, authentication, identity, authorization, open protocols, and evaluation.
NIST IR 8596, *Cybersecurity Framework Profile for Artificial Intelligence*, initial preliminary draft, December 2025	Organizes AI cybersecurity around securing AI systems, conducting AI-enabled defense, and thwarting AI-enabled attacks, including agentic use and supply-chain integrity.
NIST AI 100-2e2025, *Adversarial Machine Learning: A Taxonomy and Terminology of Attacks and Mitigations*	Provides threat terminology and a risk framework for adversarial machine-learning systems.
IETF RFC 9943, *An Architecture for Trustworthy and Transparent Digital Supply Chains*, 2026	Defines SCITT transparency services for registering signed statements and returning receipts; relevant to external witnessing of nano-ledger checkpoints and agent actions.
Coalition for Content Provenance and Authenticity, C2PA Technical Specification and AI/ML guidance	Provides signed content credentials and provenance structures for media and synthetic content; complementary to DSU object history and protected provenance payloads.
W3C PROV family of recommendations	Supplies interoperable Entity, Activity, and Agent concepts for general provenance mapping.
W3C Verifiable Credentials Data Model 2.0	Provides issuer-holder-verifier claim semantics that can be stored, transferred, and validated inside DSU-oriented workflows.
HL7 FHIR Provenance and AuditEvent	Provide clinical provenance and security-audit structures used by the LightDSU profile model.
Research Object Crate, RO-Crate Specification	Provides machine-readable packaging and metadata for FAIR research objects and reproducible workflows.
Global Alliance for Genomics and Health, Data Use Ontology	Supplies controlled terms for permitted and prohibited genomic-data uses and access decisions.

Source	Relevance to this book
ISO 8000-120 and OECD guidance on GLP data integrity	Provide enterprise master-data provenance and regulated scientific-data integrity requirements relevant to profile-based validation.

The conceptual framework should be compared with these standards rather than positioned as a replacement for them. OpenDSU contributes a protected reconstructable object, capability-derived authority, an anchor or nano-ledger for object continuity, optional SVD micro-ledgers, plural domain resolution, and a place to combine several provenance and receipt formats. External standards provide specialized claim, provenance, transparency, identity, media, software, clinical, and scientific semantics.