CONFIDENTIAL

UNITED KINGDOM INTELLECTUAL PROPERTY OFFICE

PATENT APPLICATION

Applicant

The Bitcoin Corporation Ltd

Inventor

Richard Boase

Date of Preparation

10 March 2026

Title of Invention

Signed Semantic Relationship Layer Using UTXO-Inscribed Triples with Identity-Bound Authorship and Personal Trust-Weighted Query Resolution

Field of the Invention

The present invention relates to systems and methods for expressing, storing, and querying semantic relationships between digital entities — files, identities, documents, datasets, and other addressable content — using cryptographically signed triples inscribed as unspent transaction outputs (UTXOs) on a blockchain. The invention addresses the absence of a universal, machine-queryable layer of meaning in digital infrastructure, where the relationships between things (authorship, versioning, derivation, depiction, reply, membership, endorsement) are currently trapped in private application databases, inaccessible to other applications, unverifiable by third parties, and dependent on the continued operation of the application that created them.

Background of the Invention

Problem Statement

1. **Meaning Is Trapped in Application Silos**

   The relationship between a document and its author is not expressed anywhere that a machine can independently find and verify. The fact that two files are versions of the same work is not expressed. The fact that a photograph depicts a specific person, that a message is a reply to another message, that a dataset was produced by a specific instrument, that a translation derives from an original — none of these semantic relationships exist in shared infrastructure. They exist only in the private databases of individual applications, each using its own proprietary schema, each inaccessible to every other application.

   A user who creates a document in one application, annotates it in another, and shares it via a third has created semantic relationships — authorship, annotation, distribution — that exist in three separate databases with no shared representation. If any of those applications ceases to operate, the relationships it maintained are lost. The content may survive (as dumb files), but the meaning — who created it, what it relates to, how it was modified — does not.

2. **No Universal Relationship Primitive**

   Digital infrastructure provides primitives for storing data (files), transmitting data (network protocols), and identifying locations (URLs). It provides no primitive for expressing relationships between data. The statement “Document A was authored by Person B” has no standard representation that is simultaneously: machine-readable, cryptographically signed by the asserter, verifiable without contacting a server, queryable by any application, and persistent independent of any platform.

   Every application that needs to express relationships must invent its own mechanism. Photo applications build metadata databases. Email clients build threading indices. Version control systems build commit graphs. Social platforms build follower/following tables. Each of these is a private, incompatible implementation of the same fundamental operation: asserting a typed relationship between two entities.

3. **The Semantic Web Failed for Structural Reasons**

   The W3C Semantic Web initiative (RDF, OWL, SPARQL) correctly identified that semantic relationships should be first-class, queryable data expressed as triples (subject, predicate, object). The initiative failed to achieve mainstream adoption for three structural reasons:

a. **Built on URLs that rot.** RDF triples use URIs as identifiers for subjects, predicates, and objects. When the server hosting a URI goes offline, the identifier ceases to resolve. The semantic relationships become pointers to nothing. The entire system inherits the fragility of the location-based addressing it depends upon.

b. **Required centralised ontology agreement.** For triples to be interoperable, different parties must agree on shared vocabularies (ontologies) defining the predicates. In practice, ontology agreement proved intractable. Different communities defined overlapping, conflicting vocabularies. The dream of universal interoperability collapsed into a landscape of incompatible schemas.

c. **No trust model.** In RDF, anyone can assert any triple about anything. There is no mechanism for evaluating whose assertions matter. The statement "Document X was authored by Person Y" and the statement "Document X was authored by Person Z" have equal standing. Without a trust layer, the system cannot distinguish authoritative assertions from spam, errors, or deliberate misinformation.

The core insight of the Semantic Web — that relationships should be first-class data — was correct. The foundation it was built upon — mutable URLs, centralised ontologies, no trust — was not.

4. **Relationships Cannot Be Verified**

When a platform asserts that a user authored a piece of content, or that two items are related, or that a review is genuine, the assertion is backed only by the platform's authority. There is no cryptographic proof. The platform could fabricate, alter, or delete the assertion at any time. Users have no independent means of verifying that a claimed relationship is genuine — that the person who allegedly authored a document actually signed the authorship claim, that the timestamp is accurate, that the relationship has not been silently modified.

5. **No Personal View of Meaning**

When relationships do exist in platforms, they are presented through a single, platform-determined lens. A social platform decides which relationships to surface (algorithmic feed), which to suppress (shadow banning, content moderation), and which to fabricate (promoted content presented as organic). Users cannot apply their own trust judgements to filter, weight, or evaluate relationships. The meaning layer — such as it is — is controlled by the platform, not the person.

Prior Art Limitations

RDF/Semantic Web (W3C) — Correctly identified the triple as the fundamental unit of meaning but built on mutable URLs, required centralised ontology consensus, and provided no trust model. No mainstream adoption outside academic and government data publishing.

JSON-LD — A serialisation format for linked data that is used in Schema.org markup for search engine optimisation. Embeds semantic data in web pages but inherits all URL fragility problems, has no signing mechanism, no trust model, and depends on the continued operation of the hosting web server.

Knowledge graphs (Google, Wikidata) — Centralised repositories of semantic relationships maintained by corporations or communities. Not user-controlled, not cryptographically signed, not independently verifiable. The relationships exist at the pleasure of the operator.

Git commit metadata — Expresses authorship and derivation relationships (commits, merges, tags) using content-addressed hashes and cryptographic signatures. Solves the problem for source code but is specific to the version control domain and not generalised to arbitrary semantic relationships.

GraphChain and RDF-on-blockchain research (2018–present) — Academic work has demonstrated storing RDF triples in blockchain transactions, including INSERT, DELETE, and UPDATE operations encoded in transaction data fields. GraphChain (Marum et al., 2018) proposed a blockchain-compliant distributed database exposing data with explicit RDF semantics. These approaches demonstrate the feasibility of blockchain-hosted semantic data but do not address content-addressed identifiers (they inherit URL-based RDF identifiers), do not bind assertions to self-sovereign identity chains, do not provide personal trust-weighted query resolution, and do not define an open predicate vocabulary with an aliasing mechanism for emergent convergence. They are storage mechanisms, not meaning infrastructure.

tSPARQL and trust-aware RDF queries (Hartig, 2008) — The tSPARQL language extends SPARQL with trust-weighted query semantics, associating trust values with individual RDF triples and enabling users to specify trust requirements in queries. This is the closest prior art to the trust-weighted query resolution described in the present invention. However, tSPARQL operates on traditional RDF graphs hosted on servers using URL-based identifiers — it does not use content-addressed identifiers, does not inscribe triples on a blockchain, does not bind trust to self-sovereign identity chains, and does not provide locally-stored personal trust configurations with per-predicate scoping, temporal decay rates, or configurable transitive trust depth. tSPARQL provides a query language extension; the present invention provides a complete trust-weighted semantic infrastructure built on content-addressed, identity-bound, blockchain-persistent triples.

W3C Verifiable Credentials (VC Data Model v2.0, 2025) — Verifiable Credentials provide a standard for expressing signed claims made by an issuer about a subject, using a three-party model (issuer, holder, verifier). VCs are cryptographically signed and tamper-evident. However, VCs are credential documents — structured bundles of claims issued for a specific purpose (e.g., a driver's licence, a degree certificate) — not general-purpose semantic triples. A VC asserts "Issuer X certifies that Subject Y has Property Z." The present invention asserts arbitrary typed relationships between any content-addressed entities: "Entity A [predicate] Entity B, signed by Identity C." VCs do not use content-addressed identifiers for subjects and objects (they use DIDs and URLs), are not inscribed as UTXOs on a blockchain (they are typically exchanged peer-to-peer or stored off-chain), do not provide personal trust-weighted query resolution, and do not define an open predicate vocabulary with aliasing. VCs solve credential exchange; the present invention solves universal semantic relationship persistence and querying.

Ceramic Network — Ceramic provides signed, content-addressed data streams anchored to a blockchain, using DIDs for identity and IPLD for content addressing. Ceramic streams are mutable document streams — append-only logs of changes to a document — not semantic triples expressing typed relationships between entities. Ceramic does not define a predicate vocabulary, does not provide trust-weighted query resolution where different users see different views, and does not provide a predicate aliasing mechanism. Ceramic solves mutable document synchronisation; the present invention solves semantic relationship expression and subjective querying.

IPLD (InterPlanetary Linked Data) — IPLD provides a data model for content-addressable, hash-linked data structures, unifying blockchains, Git, and other hash-linked systems into a common addressing scheme. IPLD is a data model and addressing layer, not a semantic relationship layer. It provides no mechanism for expressing typed relationships (predicates) between entities, no signing of assertions by identified authors, no trust-weighted query resolution, and no predicate vocabulary. IPLD solves content addressing; the present invention uses content addressing as one component of a semantic layer that also requires identity binding, trust configuration, and predicate semantics.

AT Protocol / Bluesky labelling system — The AT Protocol provides a "stackable moderation" architecture where independent labelling services apply labels to content, and users subscribe to labelling services of their choice, producing different views of the same content for different users. This is the closest prior art to the subjective-view principle described in the present invention. However, AT Protocol labels are binary flags (a label is either applied or not), not typed semantic relationships. Labels do not use content-addressed identifiers (they reference AT Protocol URIs). Labels are not inscribed on a blockchain. The labelling system provides no predicate vocabulary, no aliasing mechanism, and no continuous

trust degrees with decay rates and transitive depth. AT Protocol solves content moderation with user choice; the present invention solves universal semantic relationship expression with personal trust-weighted resolution.

Richardson, Agrawal, and Domingos (2003), "Trust management for the semantic web" — This foundational paper proposes trust propagation algorithms for the semantic web, including belief propagation across web-of-trust networks. The paper establishes that trust in semantic assertions should be computed from chains of trust between agents, using probabilistic models. However, Richardson et al. operate on traditional URL-based RDF graphs, require centralised trust computation (the algorithm runs over the entire graph), and do not use content-addressed identifiers, blockchain inscription, identity chains, or locally-stored per-predicate trust configurations. Richardson et al. provide a trust propagation algorithm; the present invention provides a complete infrastructure where trust is personal, local, per-predicate-scoped, temporally decaying, and applied to content-addressed identity-bound triples inscribed on a UTXO blockchain.

KERI (Key Event Receipt Infrastructure) and ACDC (Authentic Chained Data Containers) (Samuel M. Smith, 2019–present) — KERI provides a decentralised key management infrastructure using key event logs, enabling self-certifying identifiers that do not depend on any particular blockchain or ledger. ACDC extends KERI to provide chained, signed data containers for verifiable credentials. KERI/ACDC is the closest prior art to the identity-binding component of the present invention. However, KERI/ACDC focuses on credential issuance and verification (who said what about whom) using a three-party credential model, not on general-purpose semantic triple expression with an open predicate vocabulary. KERI/ACDC does not inscribe assertions as UTXOs on a blockchain (it uses its own KEL/TEL infrastructure), does not provide personal trust-weighted query resolution with per-predicate scoping and temporal decay, and does not define a predicate aliasing mechanism for emergent vocabulary convergence. KERI/ACDC solves decentralised credential management; the present invention solves universal semantic relationship expression and subjective querying.

Nanopublications and Trusty URIs (Kuhn et al., 2013–present) — Nanopublications represent the smallest publishable unit of scientific information: a single assertion (an RDF triple), together with provenance metadata and publication information, bundled into a named graph. Trusty URIs extend nanopublications with content-addressed identifiers using cryptographic hashes. Nanopublications with Trusty URIs are the closest prior art to the content-addressed signed assertion concept in the present invention. However, nanopublications are designed for scientific publishing, use traditional RDF infrastructure (SPARQL endpoints, triple stores), are not inscribed on a blockchain, do not bind assertions to self-sovereign identity chains, and do not provide personal trust-weighted query resolution

where different users see fundamentally different views of the same assertion set. Nanopublications solve scientific provenance; the present invention solves universal meaning persistence with subjective trust-weighted resolution.

OriginTrail Decentralized Knowledge Graph (DKG, 2018–present) — OriginTrail provides a decentralised knowledge graph protocol where knowledge assets (structured data including RDF) are published to a network of nodes, anchored to multiple blockchains (Ethereum, Polkadot), and discoverable via a decentralised search protocol. DKG is the closest prior art to the decentralised semantic data infrastructure described in the present invention. However, OriginTrail's knowledge graph presents a single, consensus-determined view — all participants see the same graph. OriginTrail does not provide personal trust-weighted query resolution where different users, applying different trust configurations, produce different query results from the same underlying data. OriginTrail does not bind assertions to self-sovereign identity chains (it uses wallet addresses), does not inscribe individual triples as UTXOs, does not provide per-predicate trust scoping with temporal decay, and does not define an aliasing mechanism for emergent predicate vocabulary convergence. OriginTrail solves decentralised knowledge asset management; the present invention solves subjective meaning infrastructure.

Fluree (2017–present) — Fluree is an immutable, time-ordered semantic graph database that combines JSON-LD data with blockchain-style immutability and cryptographic verification. Fluree supports semantic queries (SPARQL, GraphQL) over time-travelling data. However, Fluree is a database product — data is stored in a Fluree instance, not inscribed as UTXOs on a public blockchain. Fluree does not provide personal trust-weighted query resolution (all queries against a Fluree instance return the same results), does not bind data to self-sovereign identity chains, and does not define an open predicate vocabulary with aliasing. Fluree solves immutable semantic data storage; the present invention solves decentralised meaning persistence with subjective trust resolution.

DCoSL (Decentralized Collaboration on Semantic Linked data, 2020–present) — DCoSL proposes collaborative maintenance of semantic linked data standards using decentralised governance, enabling communities to converge on shared vocabularies without central authorities. DCoSL addresses vocabulary governance — how communities agree on predicate definitions — but does not address trust-weighted query resolution, blockchain inscription of individual triples, identity-bound signing, or temporal trust decay. DCoSL solves decentralised vocabulary governance; the present invention solves universal semantic assertion, persistence, and subjective querying.

The Applicant's prior $401 identity protocol — Provides self-sovereign on-chain identity with device-level key management and OAuth provider strands. Establishes who a person is

but does not provide a general mechanism for that person to assert signed relationships about arbitrary content.

The Applicant's prior $402 commerce protocol — Provides content addressing via UTXO inscriptions with monetary signalling. Establishes that content exists on-chain and can be economically transacted but does not provide a general mechanism for expressing semantic relationships between content items or between content and identity.

The Applicant's prior Proof of Indexing Overlay State Verification patent — Provides verification of derived state computations across competing indexer nodes but does not address the semantic layer — what relationships exist — only whether computational state is correctly derived from raw data.

No existing system combines the six properties that, together, constitute the present invention: (1) content-addressed identifiers for subjects and objects, eliminating URL rot; (2) identity-bound cryptographic signing linking every assertion to a self-sovereign on-chain identity chain; (3) UTXO inscription on a blockchain for persistence, immutability, and economic spam resistance; (4) personal trust-weighted query resolution with per-predicate scoping, continuous trust degrees, temporal decay, and configurable transitive trust depth, applied to content-addressed identity-signed triples rather than to URL-based RDF graphs; (5) an open predicate vocabulary with a trust-weighted aliasing mechanism enabling emergent convergence without centralised ontology agreement; and (6) integration across identity, commerce, compliance, and verified indexing protocols via a unified semantic layer. Individual components of this combination exist in prior art; the combination does not.

Summary of the Invention

The present invention provides a system and method for a universal signed semantic relationship layer ("the Semantic Layer") comprising:

(a) A Signed Semantic Triple — the fundamental unit of the system — comprising a subject (a content-addressed identifier for any digital entity), a predicate (a human-readable relationship type from an open, non-centralised vocabulary), an object (a content-addressed identifier for another digital entity, or a literal value), and an authorship block comprising the asserter's public key (linked to a $401 identity chain), a cryptographic signature over the triple, and a timestamp. The triple is inscribed as a UTXO on a blockchain, making it persistent, independently verifiable, and resistant to tampering or silent deletion.

(b) Content-Addressed Identifiers for All Entities — Subjects and objects in triples are identified by cryptographic hashes of their content (for files, documents, images, datasets) or by public keys (for identities). This eliminates URL rot: a triple expressing "Document [hash-

A] was authored by Identity [pubkey-B]" remains valid and resolvable regardless of whether any particular server is operational. The identifiers are mathematical facts, not location promises.

(c) An Open Predicate Vocabulary — Predicates are human-readable strings (e.g., `authored`, `replied-to`, `derived-from`, `depicts`, `translates`, `endorses`, `version-of`, `member-of`) drawn from an open vocabulary. No centralised authority controls the vocabulary. Any party may introduce new predicates. Interoperability emerges from convergence on common predicates through use, not from top-down standardisation. The system includes a predicate aliasing mechanism whereby commonly used predicates can be registered on-chain with aliases, enabling different communities to map their preferred terminology to shared semantics without requiring agreement on a single canonical vocabulary.

(d) Identity-Bound Authorship — Every triple is signed by its asserter using a private key linked to a $401 on-chain identity chain. This means: the asserter is identified (not anonymous), the assertion is cryptographically bound to a specific identity, the assertion cannot be forged (only the holder of the private key can produce a valid signature), and the assertion can be independently verified by anyone with access to the blockchain (the signature, the public key, and the triple data are all on-chain). Device-level key revocation (as provided by the $401 protocol) enables an identity to revoke a compromised device's keys without invalidating the identity itself or the triples signed by non-compromised keys.

(e) Personal Trust-Weighted Query Resolution — When a user or application queries the Semantic Layer (e.g., "Who authored Document X?" or "What documents does Person Y endorse?"), the query is resolved against the user's personal trust configuration. The trust configuration specifies: which identities the user trusts, about which predicates, to what degree (a continuous value, not binary), with what decay over time, and informed by transitive trust (trusting the trust judgements of trusted parties). Different users querying the same triples may receive different results, because the query resolver weights and filters triples according to the querying user's trust configuration. This is not a deficiency — it is the correct architecture for a world where people genuinely disagree about whose assertions are reliable.

(f) UTXO Inscription for Persistence and Spam Resistance — Each triple is inscribed as a 1-satoshi UTXO on a Bitcoin-protocol blockchain. This provides: persistence (the triple exists as long as the blockchain exists, independent of any server), immutability (once inscribed, the triple cannot be silently modified or deleted), economic spam resistance (inscribing a triple has a non-zero cost, preventing unlimited spam without requiring a centralised gatekeeper), and Merkle-provable existence (each triple has a Merkle proof linking it to a block header, enabling compact proof that the triple was inscribed at a specific time).

Detailed Description of the Invention

1. Triple Structure

1.1 Canonical Triple Format

Each signed semantic triple comprises the following fields, serialised in a canonical deterministic binary encoding:

Field	Type	Size (bytes)	Description
`version`	uint8	1	Protocol version (currently 0x01)
`subject_type`	uint8	1	0x01 = SHA-256 content hash, 0x02 = compressed public key, 0x03 = transaction ID
`subject`	bytes	32 or 33	The entity the triple is about (32 bytes for hash/txid, 33 bytes for compressed pubkey)
`predicate_length`	uint16 (big-endian)	2	Length of the predicate string in bytes
`predicate`	UTF-8 string	variable	The relationship type (e.g., “authored”, “version-of”)
`object_type`	uint8	1	0x01 = SHA-256 hash, 0x02 = compressed public key, 0x03 = transaction ID, 0x04 = UTF-8 literal, 0x05 = integer literal, 0x06 = float literal
`object_length`	uint16 (big-endian)	2	Length of the object field in bytes
`object`	bytes	variable	The related entity or value
`asserter`	bytes	33	Compressed secp256k1 public key of the asserter, linked to a $401 identity chain
`timestamp`	uint64 (big-endian)	8	Unix epoch in seconds
`signature`	bytes	64 or 65	ECDSA signature (r, s) over SHA-256 of all preceding fields, or 65 bytes with recovery flag

Canonical serialisation procedure:

1. Concatenate all fields from `version` through `timestamp` in the exact byte order specified above, with no padding or delimiters — the fixed-size and length-prefixed variable-size fields are self-delimiting.
2. Compute the SHA-256 hash of the concatenated bytes. This produces the 32-byte **signing digest**.
3. The asserter signs the signing digest using ECDSA on the secp256k1 curve (the same curve used by Bitcoin) with their private key. The signature is appended as the final field.
4. The complete serialised triple (all fields including signature) is the **canonical triple byte sequence**, used for inscription and transmission.

Any party can verify a triple by: (a) parsing the fields using the type and length prefixes, (b) re-concatenating the fields from `version` through `timestamp`, (c) computing the SHA-256 hash, (d) verifying the ECDSA signature against the `asserter` public key and the computed hash. No external service is required.

The **triple hash** — the SHA-256 hash of the complete canonical triple byte sequence including signature — serves as the content-addressed identifier for the triple itself.

1.2 Subject and Object Identification

Content-addressed entities (files, documents, images, datasets) are identified by their SHA-256 hash (`subject_type` or `object_type` = 0x01). The hash is computed over the raw bytes of the content with no additional framing or metadata. This is the same hashing algorithm used by the underlying blockchain, ensuring consistency. For a given file, any party on any machine at any time will compute the same 32-byte identifier.

Identity entities (persons, organisations, devices) are identified by their compressed secp256k1 public key (`subject_type` or `object_type` = 0x02), 33 bytes: a 0x02 or 0x03 prefix byte indicating the parity of the y-coordinate, followed by the 32-byte x-coordinate. This key must be registered in a $401 on-chain identity chain — the TripleTopicManager (Section 2.2) verifies this linkage during admission.

Transaction references (`subject_type` or `object_type` = 0x03) identify other blockchain transactions by their 32-byte transaction ID (double-SHA-256 of the serialised transaction, as standard in Bitcoin). This enables triples to reference other triples, token inscriptions, or any on-chain event.

Literal values (`object_type` 0x04, 0x05, 0x06) are permitted as objects but not as subjects. A triple may assert “Document [hash-A] has-title ‘Annual Report 2026’” where the object is a

UTF-8 literal (type 0x04). Integer literals (type 0x05) are encoded as signed 64-bit big-endian values. Float literals (type 0x06) are encoded as IEEE 754 double-precision 64-bit big-endian values. The type prefix byte ensures unambiguous parsing — a 32-byte object field is a hash only if `object_type` is 0x01, not if it is 0x04 (which would be a 32-byte UTF-8 string).

1.3 Predicate Conventions

Predicates are freeform human-readable strings. The system imposes no fixed vocabulary. However, the following core predicates are defined in the reference implementation as a starting vocabulary:

Authorship predicates: - `authored` — the subject was created by the object identity - `co-authored` — the subject was co-created by the object identity - `edited` — the subject was modified by the object identity - `signed` — the subject was cryptographically signed by the object identity

Derivation predicates: - `version-of` — the subject is a version of the object - `derived-from` — the subject was derived from the object - `translates` — the subject is a translation of the object - `summarises` — the subject is a summary of the object - `quotes` — the subject contains a quotation from the object - `replies-to` — the subject is a reply to the object

Classification predicates: - `depicts` — the subject (image/video) depicts the object (person/place/thing) - `tagged` — the subject is tagged with the object (a literal tag value) - `categorised-as` — the subject belongs to the category identified by the object - `member-of` — the subject is a member of the group/collection identified by the object

Endorsement predicates: - `endorses` — the asserter endorses the subject - `disputes` — the asserter disputes the subject - `reviews` — the subject is a review of the object - `certifies` — the asserter certifies a property of the subject

Economic predicates: - `owns` — the subject identity owns the object asset - `licensed-to` — the subject content is licensed to the object identity - `revenue-share` — the subject content has a revenue-sharing relationship with the object identity

New predicates can be introduced by anyone at any time by simply using them in a triple. If a predicate gains traction, it becomes a de facto standard through use, not through committee approval. The predicate aliasing mechanism (Section 3) enables convergence without enforcement.

2. Inscription and Storage

2.1 UTXO Inscription — Script Structure

Each signed semantic triple is inscribed on the BSV blockchain as a 1-satoshi output using a Bitcoin script envelope. The inscription uses an OP_FALSE OP_RETURN data carrier output, structured as follows:

```
OP_FALSE OP_RETURN
  <protocol_id: 4 bytes>    // 0x53545250 (ASCII "STRP" — Semantic
Triple Protocol)
  <version: 1 byte>         // 0x01
  <triple_data: variable>   // The complete canonical triple byte
sequence (Section 1.1)
```

The complete transaction structure:

Input 0: A UTXO controlled by the asserter's secp256k1 private key. The input's scriptSig contains a standard P2PKH signature and the asserter's compressed public key. This public key MUST match the `asserter` field in the triple data — the TripleTopicManager (Section 2.2) rejects transactions where these do not match, preventing one identity from inscribing triples that claim to be asserted by another.

Output 0 (inscription): The OP_FALSE OP_RETURN output containing the protocol identifier and triple data, carrying exactly 1 satoshi (the minimum dust limit on BSV).

Output 1 (change): A standard P2PKH output returning remaining funds to the asserter's address or another address they control.

The OP_FALSE OP_RETURN pattern ensures the output is provably unspendable and will not be included in the UTXO set maintained by miners, minimising storage burden on the network. However, the transaction data (including the OP_RETURN payload) is permanently recorded in the block and is retrievable by any node that maintains full block data.

The **transaction ID** (txid) — the double-SHA-256 of the serialised transaction — serves as the permanent, Merkle-provable reference for the triple. Any party can verify that the triple was inscribed at a specific block height by presenting: (a) the raw transaction, (b) the Merkle path from the transaction to the block's Merkle root, and (c) the block header. This verification requires no trusted third party.

2.2 Overlay Indexing — Data Structures and Admission

An overlay network (as defined by BRC-24) indexes inscribed triples for efficient querying. The overlay comprises two components:

TripleTopicManager — Admission logic executed when a new transaction is submitted to the overlay. The TopicManager performs the following validation steps in order:

1. **Protocol check**: Verify that Output 0 contains OP_FALSE OP_RETURN followed by the 4-byte protocol identifier 0x53545250.
2. **Version check**: Verify that the version byte is a supported version (currently only 0x01).
3. **Format parse**: Parse the canonical triple byte sequence using the type and length prefixes defined in Section 1.1. Reject if any field has an invalid type code, if the predicate length exceeds 256 bytes, if the object length exceeds 65,536 bytes, or if the total triple byte sequence exceeds the maximum inscription size (configurable, default 100,000 bytes).
4. **Signature verification**: Re-compute the signing digest (SHA-256 of all fields from `version` through `timestamp`) and verify the ECDSA signature against the `asserter` public key. Reject if verification fails.
5. **Input-asserter match**: Verify that the compressed public key in Input 0's scriptSig matches the `asserter` field in the triple. Reject if they differ.
6. **Identity chain verification**: Query the $401 identity overlay to confirm that the `asserter` public key is linked to a valid, non-revoked identity chain. Reject if the key is not found or is revoked.
7. **Admission**: If all checks pass, the triple is admitted to the overlay. The TopicManager emits an admit event containing the txid, the parsed triple fields, and the block height.

TripleLookupService — Maintains five primary indexes for efficient querying:

Index	Key	Value	Purpose
`idx_subject`	subject bytes (32 or 33)	Set of triple txids	Find all triples about a given entity
`idx_predicate`	SHA-256(predicate string)	Set of triple txids	Find all triples of a given relationship type
`idx_object`	object bytes (variable)	Set of triple txids	Find all triples pointing to a given entity
`idx_asserter`	asserter pubkey (33 bytes)	Set of triple txids	Find all triples asserted by a given identity
`idx_composite`	(subject, predicate) pair	Set of triple txids	Find specific relationships about a given entity

Each index entry stores the txid of the inscribing transaction, which serves as the pointer to the full triple data. The LookupService supports the following query operations:

- `queryBySubject(subject_bytes)` → returns all triples where the subject matches
- `queryByPredicate(predicate_string)` → returns all triples where the predicate matches
- `queryBySubjectPredicate(subject_bytes, predicate_string)` → returns all triples where both match
- `queryByAsserter(asserter_pubkey)` → returns all triples asserted by a specific identity
- `queryByObject(object_bytes)` → returns all triples pointing to a specific entity (useful for reverse lookups: “who has authored this document?” becomes queryByObject on the author’s pubkey with predicate filter)

Multiple independent nodes host the triple overlay, synchronising their raw UTXO data via GASP (Graph-Aware Sync Protocol) and verifying their derived index state via the Applicant’s prior Proof of Indexing Overlay State Verification patent. At each checkpoint interval, nodes compute a State Commitment — a Merkle root over the canonically serialised contents of all five indexes — and compare commitments across nodes. Divergent commitments trigger the Challenge-Response Dispute Protocol, ensuring that no single indexer can serve fabricated or incomplete query results without detection.

2.3 Batch Inscription — Transaction Structure

For efficiency, multiple triples may be inscribed in a single transaction. A batch inscription uses multiple OP_FALSE OP_RETURN outputs, one per triple:

```
Input 0: UTXO controlled by asserter's key

Output 0: OP_FALSE OP_RETURN <0x53545250> <0x01> <triple_1_bytes>
Output 1: OP_FALSE OP_RETURN <0x53545250> <0x01> <triple_2_bytes>
...
Output N-1: OP_FALSE OP_RETURN <0x53545250> <0x01> <triple_N_bytes>
Output N: Change output (P2PKH)
```

Each triple in the batch is individually signed by the asserter — the `signature` field in each triple covers only that triple's data, not the other triples in the batch. The TopicManager admits each triple individually, enabling per-triple querying. The batch amortises the single transaction fee (miner fee for the transaction's total byte size) across all inscribed triples, reducing per-triple cost. In the reference implementation, a batch of 100 triples of average size (approximately 200 bytes each) produces a transaction of approximately 20,500 bytes, costing approximately 10,250 satoshis at BSV's standard fee rate of 0.5 satoshis per byte — approximately 103 satoshis (less than $0.001 USD at current rates) per triple.

3. Predicate Aliasing — Mechanism and Resolution

To address the ontology fragmentation problem that undermined the Semantic Web, the system provides a predicate aliasing mechanism that enables convergence without requiring agreement.

3.1 Alias Triple Structure

An alias assertion is a standard signed semantic triple with a reserved predicate:

- **Subject**: SHA-256 hash of the UTF-8 bytes of predicate string A (e.g., SHA-256("authored") → 0x7F3A…)
- **Predicate**: `"alias-of"` (a reserved predicate in the system)
- **Object**: SHA-256 hash of the UTF-8 bytes of predicate string B (e.g., SHA-256("created-by") → 0xB2E1…)
- **Asserter**: The identity asserting the equivalence
- **Timestamp**: When the alias was asserted

The subject and object are predicate hashes (type 0x01), not the predicate strings themselves. The TripleLookupService maintains an additional index:

Index	Key	Value
`idx_alias`	predicate hash (32 bytes)	Set of {aliased predicate hash, asserter pubkey, txid} tuples

3.2 Trust-Weighted Alias Expansion

When a user submits a query with a specific predicate (e.g., “authored”), the query resolver optionally expands the predicate to include aliases:

```
function expandPredicate(user_config, predicate, now,
min_alias_trust=0.3):
    predicate_hash = SHA256(UTF8(predicate))
    aliases = idx_alias.lookup(predicate_hash)  // All alias assertions
for this predicate

    expanded_predicates = {predicate}  // Always include the original

    for each alias A in aliases:
        // Compute the querying user's trust in the alias asserter
        trust = effectiveTrust(user_config, A.asserter, "alias-of", now)
        if trust >= min_alias_trust:
            // Resolve the aliased predicate hash back to its string
            aliased_string =
lookupPredicateString(A.aliased_predicate_hash)
            if aliased_string is not null:
                expanded_predicates.add(aliased_string)

    return expanded_predicates
```

The `min_alias_trust` threshold (default 0.3) controls how readily a user accepts alias assertions. A user who trusts a standards body with degree 0.9 for the `alias-of` predicate will see that body’s aliases expanded. A user who does not trust that body will not. This is the key difference from the Semantic Web’s centralised ontology approach: alias acceptance is personal and trust-weighted, not universal.

3.3 Bidirectional and Transitive Aliases

Alias assertions are directional — `A alias-of B` does not automatically imply `B alias-of A`. For bidirectional equivalence, two alias triples must be inscribed (A→B and B→A), either

by the same asserter or by different asserters. The query resolver can be configured to treat aliases as bidirectional by default (`bidirectional_aliases: true` in the user's query configuration), reversing the lookup when only one direction has been asserted by a trusted identity.

Transitive aliases (A alias-of B, B alias-of C, therefore A alias-of C) are supported up to a configurable depth (default 2) to prevent unbounded expansion. At each transitive step, the trust threshold is tightened by a factor of 1.5× (e.g., if the base threshold is 0.3, the second hop requires 0.45).

This mechanism allows different communities — scientists, lawyers, artists, engineers — to use their preferred terminology while maintaining cross-community discoverability. Convergence is emergent, not mandated.

4. Trust-Weighted Query Resolution

4.1 Trust Configuration — Data Structure

Each user maintains a personal trust configuration stored as a local JSON file on the user's device. The configuration is an array of trust assertion objects:

```
{
  "version": 1,
  "owner": "<user's compressed pubkey, 66 hex chars>",
  "created": "<unix timestamp>",
  "assertions": [
    {
      "trusted_identity": "<compressed pubkey, 66 hex chars>",
      "predicate_scope": "authored",
      "trust_degree": 0.9,
      "decay_rate": 0.01,
      "decay_unit": "day",
      "last_reinforced": 1741564800,
      "transitive": true,
      "transitive_depth": 2,
      "transitive_attenuation": 0.5
    }
  ]
}
```

Field	Type	Constraints	Description
`trusted_identity`	Hex string (66 chars)	Must be a valid compressed secp256k1 pubkey	The identity being trusted
`predicate_scope`	String or `"*"`	UTF-8, max 256 bytes; `"*"` means all predicates	Which predicates this trust applies to
`trust_degree`	Float	0.0 ≤ value ≤ 1.0	The degree of trust at last reinforcement
`decay_rate`	Float	0.0 ≤ value ≤ 1.0	Proportion of trust lost per `decay_unit` without reinforcement
`decay_unit`	String	One of: `"hour"`, `"day"`, `"week"`, `"month"`, `"year"`	Time unit for decay calculation
`last_reinforced`	uint64	Unix epoch (seconds)	When the user last explicitly confirmed this trust
`transitive`	Boolean	—	Whether to extend trust to identities trusted by this identity
`transitive_depth`	Integer	0 ≤ value ≤ 10	Maximum hops of transitive trust traversal
`transitive_attenuation`	Float	0.0 < value < 1.0	Multiplier applied to trust degree at each transitive hop

The trust configuration file is encrypted at rest using a key derived from the user's device credentials (e.g., AES-256-GCM with a key derived from the user's master private key via HKDF). It is never transmitted to any server unless the user explicitly exports it. This is consistent with the local-first principle.

4.2 Trust Computation Algorithm

The effective trust degree for a given asserter identity, from the perspective of the querying user, for a given predicate, at a given time, is computed as follows:

Step 1 — Direct trust lookup with temporal decay:

```
function directTrust(user_config, asserter_pubkey, predicate, now):
    for each assertion A in user_config.assertions:
        if A.trusted_identity == asserter_pubkey:
            if A.predicate_scope == "*" or A.predicate_scope ==
predicate:
                elapsed = now - A.last_reinforced
                decay_units = elapsed / seconds_per(A.decay_unit)
                effective_trust = A.trust_degree * (1.0 - A.decay_rate)
^ decay_units
                if effective_trust < 0.001:
                    return 0.0  // Below threshold, treat as untrusted
                return effective_trust
    return 0.0  // No matching assertion found
```

The decay formula `trust_degree × (1 - decay_rate) ^ elapsed_units` produces exponential decay. A trust degree of 0.9 with a decay rate of 0.01 per day decays to approximately 0.86 after 5 days, 0.67 after 30 days, and 0.49 after 60 days without reinforcement. The user reinforces trust by updating the `last_reinforced` timestamp, resetting the decay. This models the real-world principle that trust requires ongoing confirmation — unexamined trust should weaken over time.

Step 2 — Transitive trust expansion:

```
function effectiveTrust(user_config, asserter_pubkey, predicate, now,
depth=0, visited={}):
    // Check direct trust first
    direct = directTrust(user_config, asserter_pubkey, predicate, now)
    if direct > 0.0:
        return direct

    // If no direct trust, attempt transitive expansion
    if depth >= max_transitive_depth(user_config):
        return 0.0

    visited = visited U {asserter_pubkey}
    max_transitive = 0.0

    for each assertion A in user_config.assertions:
        if A.transitive == true and A.trusted_identity not in visited:
            // Compute direct trust to the intermediary
            intermediary_trust = directTrust(user_config,
A.trusted_identity, predicate, now)
            if intermediary_trust == 0.0:
                continue

            // Query the Semantic Layer for triples where the
intermediary
            // asserts trust in the target asserter
            // Specifically: triples with predicate "trusts" or
"endorses"
            // where asserter == intermediary and object ==
asserter_pubkey
            intermediary_endorsements = queryByAsserterAndObject(
                A.trusted_identity, asserter_pubkey, ["trusts",
"endorses"]
            )
            if intermediary_endorsements is empty:
                continue

            // Attenuate trust: intermediary's trust × attenuation
factor
            transitive_trust = intermediary_trust *
A.transitive_attenuation
            // Recurse for deeper transitive chains
            deeper = effectiveTrust(
                user_config, asserter_pubkey, predicate, now,
                depth + 1, visited
            )
            transitive_trust = max(transitive_trust, deeper)
```

```
            max_transitive = max(max_transitive, transitive_trust)

    return max_transitive
```

The transitive trust algorithm performs a bounded depth-first search through the trust graph. At each hop, the trust degree is multiplied by the `transitive_attenuation` factor (typically 0.5), producing rapid attenuation: at depth 1, trust is halved; at depth 2, quartered; at depth 3, one-eighth. The `visited` set prevents cycles. The maximum depth is bounded by the user's configuration (default 2, maximum 10). The algorithm terminates in O(N × D) time where N is the number of direct trust assertions and D is the maximum depth.

Step 3 — Query resolution with trust weighting:

```
function resolveQuery(user_config, query, now,
min_trust_threshold=0.05):
    raw_results = lookupService.query(query)  // Returns all matching
triples
    weighted_results = []

    for each triple T in raw_results:
        trust = effectiveTrust(
            user_config, T.asserter, T.predicate, now
        )
        if trust >= min_trust_threshold:
            weighted_results.append({
                triple: T,
                trust_weight: trust,
                asserter_identity: resolve_identity_chain(T.asserter)
            })

    // Sort by trust weight descending
    weighted_results.sort(key=trust_weight, descending=true)
    return weighted_results
```

The `min_trust_threshold` parameter (default 0.05, configurable by the user) filters out triples from identities with negligible trust, reducing noise. The result is an ordered list of triples ranked by the querying user's personal trust weights.

4.3 Subjective Views — Worked Example

Consider a document with hash `0xABCD...` and two conflicting authorship claims:

- Triple T1: `[0xABCD...] authored [PubKey-Alice]`, asserted by PubKey-Alice (self-assertion), timestamp 1741500000

- Triple T2: `[0xABCD...] authored [PubKey-Bob]`, asserted by PubKey-Carol (third-party assertion), timestamp 1741510000

User X's trust configuration: trusts PubKey-Alice with degree 0.8 for predicate "authored", trusts PubKey-Carol with degree 0.3 for predicate "authored".

User Y's trust configuration: trusts PubKey-Alice with degree 0.2 for predicate "authored", trusts PubKey-Carol with degree 0.95 for predicate "authored".

User X's query result: T1 (weight 0.8), T2 (weight 0.3) — Alice's self-assertion ranks first.
User Y's query result: T2 (weight 0.95), T1 (weight 0.2) — Carol's third-party assertion ranks first.

The same triples, the same query, different results — because the users have different trust configurations. Neither result is "correct" in an absolute sense. Each reflects the querying user's own trust judgements applied to cryptographically verified assertions. This is the core architectural principle: meaning is subjective, and an honest system must reflect this rather than imposing a single platform-determined view.

5. Device-Level Key Revocation

The Semantic Layer integrates with the $401 identity protocol's key management to handle device compromise without destroying an identity's entire assertion history.

5.1 Revocation Triple Structure

When a device is compromised, the identity owner inscribes a revocation triple:

- **Subject**: Compressed public key of the compromised device (type 0x02, 33 bytes)
- **Predicate**: `"revoked-by"` (a reserved predicate)
- **Object**: Compressed public key of the identity's root key or another non-compromised device key (type 0x02, 33 bytes)
- **Asserter**: The identity's root key or another non-compromised device key (MUST NOT be the compromised key itself)
- **Timestamp**: When the revocation is inscribed

The TripleTopicManager validates revocation triples with an additional check: it queries the $401 identity overlay to confirm that both the compromised key (subject) and the signing key (asserter) belong to the same identity chain, and that the signing key is not itself revoked. This prevents an attacker who has compromised one device key from revoking the identity's other keys.

5.2 Index Annotation Process

Upon admitting a revocation triple, the TripleTopicManager triggers a batch annotation in the LookupService:

```
function processRevocation(revoked_pubkey, revocation_txid,
revocation_timestamp):
    // Find all triples asserted by the revoked key
    affected_triples = idx_asserter.lookup(revoked_pubkey)

    for each txid in affected_triples:
        triple = loadTriple(txid)
        // Add revocation annotation to the index entry
        idx_revocation_status[txid] = {
            status: "SIGNED_BY_REVOKED_KEY",
            revocation_txid: revocation_txid,
            revocation_timestamp: revocation_timestamp,
            original_assertion_timestamp: triple.timestamp
        }
```

The `idx_revocation_status` index maps triple txids to their revocation annotations. This is a derived state index — its correctness is verified across competing nodes via the Proof of Indexing mechanism alongside the five primary indexes.

5.3 Trust Weight Adjustment for Revoked-Key Triples

The query resolver adjusts trust weights for triples signed by revoked keys:

```
function adjustForRevocation(triple_txid, base_trust_weight):
    revocation = idx_revocation_status.lookup(triple_txid)
    if revocation is null:
        return base_trust_weight  // No revocation, use base weight

    if revocation.status == "SIGNED_BY_REVOKED_KEY":
        // Triples asserted BEFORE the revocation are likely genuine
        // (the key was valid when they were signed)
        // Triples asserted AFTER the revocation are suspect
        // Apply a configurable penalty factor
        time_since_revocation = revocation.revocation_timestamp -
triple.timestamp
        if time_since_revocation > 0:
            // Triple was signed before revocation — apply mild penalty
            return base_trust_weight * 0.7  // Configurable:
revoked_key_pre_penalty
        else:
            // Triple was signed after revocation — highly suspect
            return base_trust_weight * 0.1  // Configurable:
revoked_key_post_penalty

    return base_trust_weight
```

Triples are never deleted from the blockchain. They remain as immutable historical records. The revocation affects only how the query resolver weights them. A forensic investigator or auditor can always access the raw, unfiltered triple set. The annotation-not-deletion model preserves the evidentiary record while protecting users from relying on assertions made by compromised keys.

5.4 Re-Assertion After Revocation

If the triples signed by a compromised key were genuine (the key was compromised after the triples were created), the identity owner can re-assert them by inscribing new triples with identical content (same subject, predicate, object) signed by a non-compromised key. The new triples receive full trust weight. The old triples remain on-chain as corroborating historical evidence. This model handles the common scenario where a device is stolen after being used legitimately for months — the legitimate assertions survive via re-assertion, while any fraudulent assertions made after the theft are penalised by the revocation annotation.

6. Integration with the Protocol Stack

The Semantic Layer integrates with the Applicant’s existing protocol stack:

$401 (Identity): Every triple asserter is identified by a $401 on-chain identity. Trust configurations reference $401 identities. Key revocation uses $401 device key management. The Semantic Layer cannot function without identity — unsigned assertions are meaningless.

$402 (Commerce): Content addressed by $402 tokens can be the subject or object of semantic triples. The relationship "Token [tx-A] represents content [hash-B]" is itself a semantic triple. Revenue events from $402 transactions can trigger automated triple inscription (e.g., "Identity [key-X] purchased Token [tx-Y] at timestamp [T]"), creating an auditable semantic record of economic activity.

$403 (Securities): Securities tokens requiring KYC can use semantic triples to express compliance relationships: "Identity [key-X] certified-as accredited-investor by Identity [key-Y]". The $403 layer can query these triples to evaluate access conditions.

Proof of Indexing: Triple overlay indexing constitutes verifiable work that can be committed to the Proof of Indexing mining mechanism. Nodes that index triples earn $402 tokens. The correctness of the triple index is verified by the POI Overlay State Verification system.

ClawMiner (Local Hardware): The user's trust configuration is stored on their ClawMiner device. Triples asserting relationships about the user's own content are signed by keys held in the ClawMiner's secure element. The device operates as the user's local semantic authority — signing assertions, storing trust configurations, and resolving queries according to the user's personal trust weights.

7. Application Scenarios

7.1 Document Authorship

A writer creates a document and inscribes a triple: `[document-hash] authored [writer-pubkey]`. Any application — a reader, a search engine, a citation tool — can query the Semantic Layer and discover who authored the document, verified by the writer's cryptographic signature. The relationship persists regardless of which application the document was created in, which platform it is hosted on, or whether any specific server remains operational.

7.2 Photograph Provenance

A photographer takes a photograph. Their device (ClawMiner) automatically inscribes: `[photo-hash] authored [photographer-pubkey]`, `[photo-hash] captured-at [timestamp]`, `[photo-hash] depicts [subject-pubkey]` (if the subject has consented and is identified). Any downstream user of the photograph can verify its provenance. AI-

generated images cannot produce a valid photographer signature. The semantic record is the provenance chain.

7.3 Version Tracking

A team edits a shared document. Each version is hashed. When a new version is saved, a triple is inscribed: `[new-hash] version-of [previous-hash]`, `[new-hash] edited [editor-pubkey]`. The complete version history is a chain of semantic triples, queryable by any application, not locked in a single version control system.

7.4 Content Moderation Without Central Authority

A community moderator flags content as violating community standards: `[content-hash] flagged-as harmful-content` signed by the moderator. Users who trust that moderator see the flag. Users who do not trust that moderator do not. No central authority decides for everyone. Each user's view of content reflects their own trust network. This is subjective moderation — the only kind that respects both free expression and community standards simultaneously.

7.5 Academic Citation

A researcher cites a source: `[paper-hash] cites [source-hash]`. The citation is cryptographically signed by the researcher's identity. Citation networks become a queryable semantic graph, independent of any journal, database, or platform. Impact metrics can be computed by any party from the public triple data, with trust weighting determining which citations count.

Brief Description of Drawings

- **Figure 1** — Signed semantic triple structure showing subject (content hash), predicate (relationship type), object (content hash or literal), asserter (public key), timestamp, and signature, with arrows indicating the cryptographic signing process.
- **Figure 2** — Content-addressed identification comparison: URLs (mutable, server-dependent, decaying) versus content hashes (immutable, self-verifying, permanent), showing how triples built on content hashes survive server shutdowns.
- **Figure 3** — UTXO inscription process: asserter constructs triple → serialises canonically → signs with private key → inscribes as blockchain UTXO → TopicManager admits to overlay → LookupService indexes for querying.

- **Figure 4** — Trust-weighted query resolution flow: user submits query → LookupService returns raw triples → query resolver applies user's trust configuration → filters by trusted identities → applies transitive trust expansion → returns trust-weighted results.
- **Figure 5** — Subjective views diagram: same set of triples viewed through two different trust configurations, producing two different result sets — demonstrating that meaning is personal, not universal.
- **Figure 6** — Predicate aliasing: Community A uses `authored`, Community B uses `created-by`, alias triples link them, query expansion returns results for both.
- **Figure 7** — Device-level key revocation: identity with three device keys, one revoked, showing that triples signed by the revoked key are annotated but not deleted, while the identity and its other keys remain valid.
- **Figure 8** — Protocol stack integration: $401 (identity) providing asserter authentication, $402 (commerce) providing content addressing, $403 (securities) consuming compliance triples, POI providing index verification, Semantic Layer providing the relationship graph connecting all layers.
- **Figure 9** — Application scenarios: document authorship, photograph provenance, version tracking, and content moderation — each showing the relevant triple(s) and query resolution path.

Initial Claims

Note: These claims are provided in sketch form for the purposes of establishing a priority date. Formal claims will be drafted and filed within 12 months in accordance with UKIPO rules.

Claim 1 — Identity-Bound Content-Addressed Semantic Triple Inscribed as Blockchain UTXO

A method for expressing and persisting semantic relationships between digital entities using content-addressed identifiers and self-sovereign identity binding, comprising:

a. constructing a semantic triple comprising a subject identifier (a cryptographic content hash of the entity's raw bytes, or a public key registered in a self-sovereign on-chain identity chain), a predicate (a human-readable relationship type drawn from an open, non-centralised vocabulary), and an object identifier (a cryptographic content hash, a

public key registered in a self-sovereign on-chain identity chain, or a typed literal value);

b. the asserter cryptographically signing the triple using a private key linked to a self-sovereign on-chain identity chain, such that the assertion is permanently bound to a verified identity rather than to an anonymous key or a platform-issued account;

c. inscribing the signed triple as an unspent transaction output (UTXO) on a blockchain, creating a persistent, immutable, Merkle-provable record of the semantic assertion that incurs a non-zero economic cost per inscription, providing spam resistance without a centralised gatekeeper;

d. an overlay network admitting the inscribed triple via a TopicManager that validates the canonical format and verifies the asserter's signature and identity chain linkage, and indexing the triple via a LookupService queryable by subject, predicate, object, asserter, and combinations thereof;

wherein subjects and objects are identified by content-addressed hashes or self-sovereign public keys rather than by mutable location-based addresses, such that the semantic relationship remains valid and verifiable regardless of whether any server, platform, or domain continues to operate; and wherein the combination of content-addressed identification, self-sovereign identity binding, UTXO inscription, and overlay indexing distinguishes the method from prior art systems that store RDF triples in blockchain transactions using URL-based identifiers without identity binding, trust-weighted resolution, or overlay verification.

Claim 2 — Personal Trust-Weighted Query Resolution over Content-Addressed Identity-Bound Semantic Triples

A method for resolving queries against a corpus of content-addressed, identity-bound, blockchain-inscribed semantic triples according to a querying user's locally-stored personal trust configuration, comprising:

a. receiving a query specifying one or more of: a subject (content hash or public key), a predicate, an object (content hash, public key, or literal), and an asserter (public key linked to a self-sovereign identity chain);

b. retrieving all matching triples from a verified overlay index maintained by competing indexer nodes whose correctness is established via a Proof of Indexing consensus mechanism;

c. applying the querying user's personal trust configuration, stored locally on the user's own device and never transmitted to a central server, the configuration comprising trust

assertions specifying: trusted identities (by public key), per-predicate scoping (trusting an identity about specific relationship types but not others), continuous trust degrees (a value between 0.0 and 1.0, not binary), temporal decay rates (trust diminishing over time without reinforcement), and configurable transitive trust parameters (extending trust to identities trusted by directly trusted identities, with attenuating trust degree per hop and a maximum traversal depth);

d. filtering and weighting the retrieved triples according to the trust configuration, such that triples asserted by highly trusted identities are weighted more heavily, triples asserted by less trusted identities are weighted proportionally, and triples asserted by identities below a configurable trust threshold are excluded;

e. returning the trust-weighted result set to the querying user;

wherein different users querying the same set of blockchain-inscribed triples receive different results reflecting their individual, locally-stored trust configurations; wherein the trust-weighted resolution operates over content-addressed, identity-bound triples rather than over URL-based RDF graphs as in prior art trust-aware query languages; and wherein no central authority, platform algorithm, or labelling service determines the canonical interpretation of the semantic data — the interpretation is computed from the user's own trust judgements applied to cryptographically verified assertions.

Claim 3 — Content-Addressed Identity-Bound Semantic Relationships with Overlay Verification

A system for maintaining verifiable semantic relationships between digital entities using content-addressed identifiers, self-sovereign identity binding, and verified overlay indexing, comprising:

a. identifying digital entities (files, documents, images, datasets) by the cryptographic hash of their content, rather than by location-based addresses, such that the identifier is a mathematical property of the entity computable by any party without contacting a server;

b. identifying persons and organisations by their public keys as registered in self-sovereign on-chain identity chains, rather than by platform-issued usernames or account identifiers;

c. expressing typed relationships between entities as signed semantic triples where subjects and objects are content-addressed, predicates are drawn from an open vocabulary, and each triple is cryptographically signed by the asserter's identity-chain-linked key;

d. persisting the signed triples as UTXOs on a blockchain, providing Merkle-provable existence timestamps and economic spam resistance;

e. indexing the persisted triples via an overlay network whose derived state correctness is verified by competing indexer nodes through a Proof of Indexing consensus mechanism;

wherein the semantic relationships remain valid and verifiable regardless of whether any particular server, platform, or domain continues to operate; wherein the combination of content-addressed identification and self-sovereign identity binding distinguishes the system from prior art content-addressed data models (such as IPLD) that provide addressing without semantic predicates, identity binding, or trust-weighted querying; and wherein the overlay verification ensures that query results are trustworthy without requiring trust in any single indexing provider.

Claim 4 — Open Predicate Vocabulary with Emergent Convergence via Aliasing

A method for enabling interoperable semantic relationships without requiring centralised vocabulary agreement, comprising:

a. permitting any party to use any human-readable string as a predicate in a signed semantic triple, without requiring registration with or approval from a central authority;

b. providing an aliasing mechanism whereby any party may inscribe an alias assertion linking two predicates as semantically equivalent;

c. alias assertions being themselves signed semantic triples, subject to the same trust-weighted query resolution as all other triples;

d. a query expansion mechanism whereby queries for triples with a given predicate may optionally be expanded to include triples with aliased predicates, according to the querying user's trust configuration;

wherein vocabulary convergence emerges from community usage patterns and trusted alias assertions rather than from top-down standardisation, and wherein different communities may use different terminology while maintaining cross-community discoverability.

Claim 5 — Device-Level Key Revocation for Semantic Assertions

A method for managing the validity of semantic assertions when a signing key is compromised, comprising:

a. an identity owner inscribing a revocation triple identifying a compromised device key, signed by the identity's root key or another non-compromised device key;

b. an overlay index processing the revocation and annotating all triples previously signed by the revoked key with a revocation status;

c. the annotated triples remaining on the blockchain as historical records, not deleted;

d. query resolvers treating triples signed by revoked keys with reduced trust weight or requiring additional corroboration;

wherein the identity itself survives the revocation, other device keys remain valid, and the granularity of revocation is at the device level rather than the identity level.

Claim 6 — Integrated Semantic Layer Across Identity, Commerce, and Compliance Protocols

A system integrating a signed semantic triple layer with blockchain-based identity, commerce, and compliance protocols, comprising:

a. a self-sovereign identity protocol providing asserter authentication and device-level key management for triple signing;

b. a content commerce protocol providing content-addressed tokens that serve as subjects and objects in semantic triples;

c. a compliance protocol consuming semantic triples expressing certification and accreditation relationships to evaluate access conditions for regulated tokens;

d. a Proof of Indexing verification system ensuring the correctness of the semantic triple index across competing overlay nodes;

e. a local hardware device storing the user's trust configuration and signing triples using keys held in a secure element;

wherein the semantic layer provides the connective tissue between identity, content, economics, and compliance, enabling machine-queryable relationships across the protocol stack that are cryptographically signed, personally trust-weighted, and persistent on the blockchain.

Abstract

A system and method for expressing, persisting, and querying semantic relationships between digital entities using cryptographically signed triples inscribed as unspent transaction outputs (UTXOs) on a blockchain. Each triple comprises a subject (identified by content hash or public key), a predicate (a human-readable relationship type), an object (identified by content hash, public key, or literal value), and an authorship block (the asserter's public key,

cryptographic signature, and timestamp). Content-addressed identifiers eliminate the URL rot that undermined the Semantic Web. Cryptographic signing enables independent verification without contacting a server. UTXO inscription provides persistence, immutability, and economic spam resistance. An open predicate vocabulary with an aliasing mechanism enables interoperability without centralised ontology agreement. Query resolution applies the querying user's personal trust configuration — specifying trusted identities, predicate scopes, trust degrees, decay rates, and transitive trust parameters — so that different users may receive different views of the same semantic data, reflecting their individual trust judgements. The system integrates with self-sovereign identity protocols (for asserter authentication and device-level key revocation), content commerce protocols (for content addressing), and compliance protocols (for certification relationships), providing the missing meaning layer in digital infrastructure where relationships between content, identity, and economic activity are first-class, machine-queryable, cryptographically verifiable, and personally interpretable.

Document prepared for UKIPO filing. Priority date to be established upon submission. Applicant: The Bitcoin Corporation Ltd Inventor: Richard Boase Date of preparation: 10 March 2026