Consensix Labs

Cross-Chain Asset Provenance Registry

A Protocol for Verifiable Chain-of-Custody Records Spanning Multiple Blockchains, with Proof-of-Concept Implementations on Ethereum and IOTA

Table of Contents

Executive Summary

Provenance – the documented history of ownership, authentication, and condition changes for a valuable asset – underpins trust in markets for art, luxury goods, vintage vehicles, rare instruments, and real estate. Today, provenance records are fragmented across filing cabinets, proprietary databases, and email threads. Verification is slow, expensive, and routinely defeated by forgery.

This paper presents a protocol for building provenance records as chains of cryptographically signed attestations anchored on public blockchains. Each attestation – an origin declaration, a transfer of custody, an authentication report, or a significant event – is a signed data structure whose SHA-256 hash is recorded on-chain. The key innovation is that a single asset's provenance chain can span multiple blockchains: different parties anchor their attestations on whichever chain they prefer, and the chain is stitched together by content hashes rather than on-chain pointers. A verification process walks the chain from the most recent attestation back to the origin, checking each link's on-chain status across all relevant chains, and detects any gaps or forgeries.

The proof of concept implements the protocol on two architecturally different blockchains – Ethereum (using Solidity) and IOTA Rebased (using Move). A demonstration scenario traces a painting through five parties – artist, gallery, appraiser, collector, and restorer – with attestations alternating between the two chains. The entire demonstration uses shell scripts and curl against live local blockchain networks, keeping the implementation minimal while genuinely exercising cross-chain operations. A forgery detection scenario demonstrates that inserting a fake attestation with an unregistered parent hash produces a detectable break in the provenance chain.

1. Introduction

The Problem

A painting moves through an artist's studio, a gallery, a private collector, an auction house, and a museum. At each step, paperwork is generated: certificates of authenticity, bills of sale, condition reports, insurance appraisals, conservation records. This paperwork constitutes the painting's **provenance** – the documented chain of custody and authentication that determines whether the work is genuine, who owns it, and what has happened to it over time.

The problem is that this documentation lives in filing cabinets, proprietary databases, email attachments, and sometimes nowhere at all. When a prospective buyer wants to verify a painting's provenance, they hire researchers who spend weeks tracing paper trails across institutions and jurisdictions. Gaps in the record are common. Forgery is rampant – the art market loses an estimated billions of dollars annually to fraud, much of it enabled by the difficulty of verifying provenance.

The same pattern applies far beyond fine art. Luxury goods (watches, handbags, wine), vintage vehicles, rare musical instruments, real estate titles, and intellectual property all depend on documented chains of custody. In every case, the documentation is fragmented, verification is manual, and the cost of fraud is borne by buyers who cannot efficiently distinguish genuine records from fabricated ones.

The Opportunity

Blockchain technology offers a natural foundation for provenance records: an immutable, timestamped, publicly auditable ledger. But existing blockchain-based provenance approaches focus on the wrong layer. Minting an NFT for a physical asset does not prove ownership – it proves that someone minted an NFT. Linking a token to a physical object requires trust in the party that created the link, and that trust is precisely what provenance is supposed to establish.

What is needed is not a token that represents an asset, but a system where each party in the chain of custody creates a cryptographically signed attestation about the asset – and the sequence of these attestations, verified independently, forms the provenance record. The blockchain's role is not to store the attestations themselves (which may be large and contain private details), but to anchor their hashes, creating a tamper-proof record that each attestation existed at a specific point in time and has not been altered since.

This paper presents such a system, with a specific focus on the **cross-chain** problem: in a world with multiple blockchains, different parties will anchor their attestations on different chains. An artist might use IOTA. A gallery might prefer Ethereum. An auction house might use a different network entirely. A practical provenance protocol must accommodate this reality, allowing a single asset's provenance chain to span multiple blockchains seamlessly.

Scope and Approach

This paper defines a protocol for cross-chain asset provenance and demonstrates it with a minimal but functional proof of concept. The PoC implements smart contracts on Ethereum and IOTA, and includes shell scripts that deploy the contracts, register attestations on both chains, walk the provenance chain across chains, and detect forgery attempts.

The approach is intentionally lighter than a full application. There is no CLI framework, no web interface, and no JWT signing at runtime. Attestation hashes are pre-computed, and the demonstration uses hardcoded sample values. The goal is to establish a credible protocol design with working on-chain code, not to build production

tooling. The protocol is designed so that a full PoC with runtime attestation signing, off-chain storage, and a user interface can be built as a future project without changing the on-chain contracts or the verification logic.

2. Background

Asset Provenance

Provenance, in the context of valuable assets, refers to the documented history of an object's ownership, location, condition, and authentication. For a painting, provenance might include the artist's original bill of sale, gallery exhibition records, auction house lot descriptions, insurance appraisals, condition reports from conservators, and certificates of authenticity from recognized experts. For a luxury watch, it might include the manufacturer's warranty card, service records, and a chain of receipts from successive owners.

The value of provenance is both cultural and economic. A painting with an unbroken chain of documented ownership from the artist's studio to the present day commands a premium over one with gaps in its record. A watch with complete service history from authorized dealers is worth more than an identical model without documentation. In regulated markets like real estate, provenance (in the form of a title chain) is a legal requirement for transfer of ownership.

The fundamental challenge is that provenance is cumulative and distributed. No single party holds the complete record. Each participant in an asset's history contributes their piece – a sale receipt, an authentication report, a restoration record – and the complete provenance is the sum of all these pieces, often held by different people and organizations across different countries and decades.

Existing Approaches

Several projects have attempted to bring provenance tracking to blockchain technology. The following table summarizes the landscape.

Approach	Type	Chain Support	Data Model	Provenance Chain	Status
Everledger	Platform (proprietary)	Private (Hyperledger)	Proprietary	Single platform	Closed (2023)
Provenance Proof	Platform (gemstones)	Private (via Everledger)	Proprietary	Single platform	Active
IBM Food Trust	Platform (supply chain)	Private (Hyperledger Fabric)	Proprietary	Single platform	Winding down
EAS (Ethereum Attestation Service)	Protocol (attestations)	EVM only	Schema-based	No chain linking	Active
NFT-based approaches	Various	Single chain per project	Token-based	Token transfers	Various
W3C PROV	Standard (data model)	None (not blockchain)	RDF/OWL ontology	Activity-based	W3C Recommendation
This protocol	Protocol (provenance)	Multi-chain	Attestation chain	Cross-chain linked	This paper

Everledger was the most prominent blockchain provenance platform, tracking diamonds, gemstones, wine, and luxury goods on a private Hyperledger-based blockchain. Founded in 2015 and backed by Tencent, Fidelity, and government grants totaling over $27 million, Everledger went into administration in 2023 after a funding round collapsed. Crunchbase lists it as permanently closed. Its failure is instructive: even a well-funded, commercially-focused provenance platform could not achieve sustainability. The proprietary, permissioned architecture created vendor lock-in, and the provenance data was tied to a single platform that no longer exists. This underscores the case for open, standards-based protocols where provenance records are not dependent on any single company's continued operation.

Provenance Proof Blockchain, operated by the House of Gubelin for colored gemstones, has tracked over 500,000 gemstones across 50+ countries. It was built on Everledger's platform, which creates a notable dependency risk given Everledger's closure. The long-term viability of provenance records created through this system is unclear.

IBM Food Trust was a Hyperledger Fabric-based supply chain traceability platform for food products. IBM's broader blockchain division was significantly reduced starting in 2021, with the blockchain team reportedly cut to near-zero after missing revenue targets. The TradeLens shipping blockchain (a joint venture with Maersk) was shut down in early 2023 after failing to achieve commercial viability. IBM Blockchain Platform software reached end of support in April 2023. While IBM Food Trust's website remains accessible, IBM's retreat from enterprise blockchain is well-documented.

Ethereum Attestation Service (EAS) is the closest technical comparison to the protocol presented here. EAS provides an open, permissionless infrastructure for making on-chain or off-chain attestations using custom schemas. It operates on EVM-compatible chains and is actively maintained as an open-source public good. The key difference is that EAS is a general-purpose attestation primitive – it does not define provenance-specific attestation types, does not enforce chain linking between attestations, and does not address cross-chain verification. Our protocol builds on similar principles (signed attestations, on-chain hash anchoring) but adds the domain-specific structure of provenance chains and the cross-chain linking mechanism.

NFT-based approaches use non-fungible tokens as ownership proxies for physical assets. The fundamental limitation is the "oracle problem" in reverse: minting a token does not establish a connection to a physical asset – it only proves that a token was minted. The token's value as a provenance record depends entirely on trust in the minting party. Our protocol takes a different approach: attestations are not ownership tokens but signed statements by identified parties, and their value comes from the cumulative weight of multiple independent attestations forming a verifiable chain.

W3C PROV is a family of W3C Recommendations (published 2013) that define a data model for provenance interchange. PROV models provenance around entities, activities, and agents, and provides RDF/OWL serializations for interoperability. It is a mature standard used primarily in scientific data management and digital preservation. PROV is complementary to our protocol – a future extension could map our attestation types to PROV concepts for interoperability with existing provenance systems – but PROV itself does not address blockchain anchoring, cross-chain verification, or the specific needs of physical asset provenance.

EU Digital Product Passport

The European Union's **Ecodesign for Sustainable Products Regulation (ESPR)**, which entered into force in July 2024, introduces the **Digital Product Passport (DPP)** – a mandatory digital record containing lifecycle, sustainability, and traceability information for products sold in the EU market.

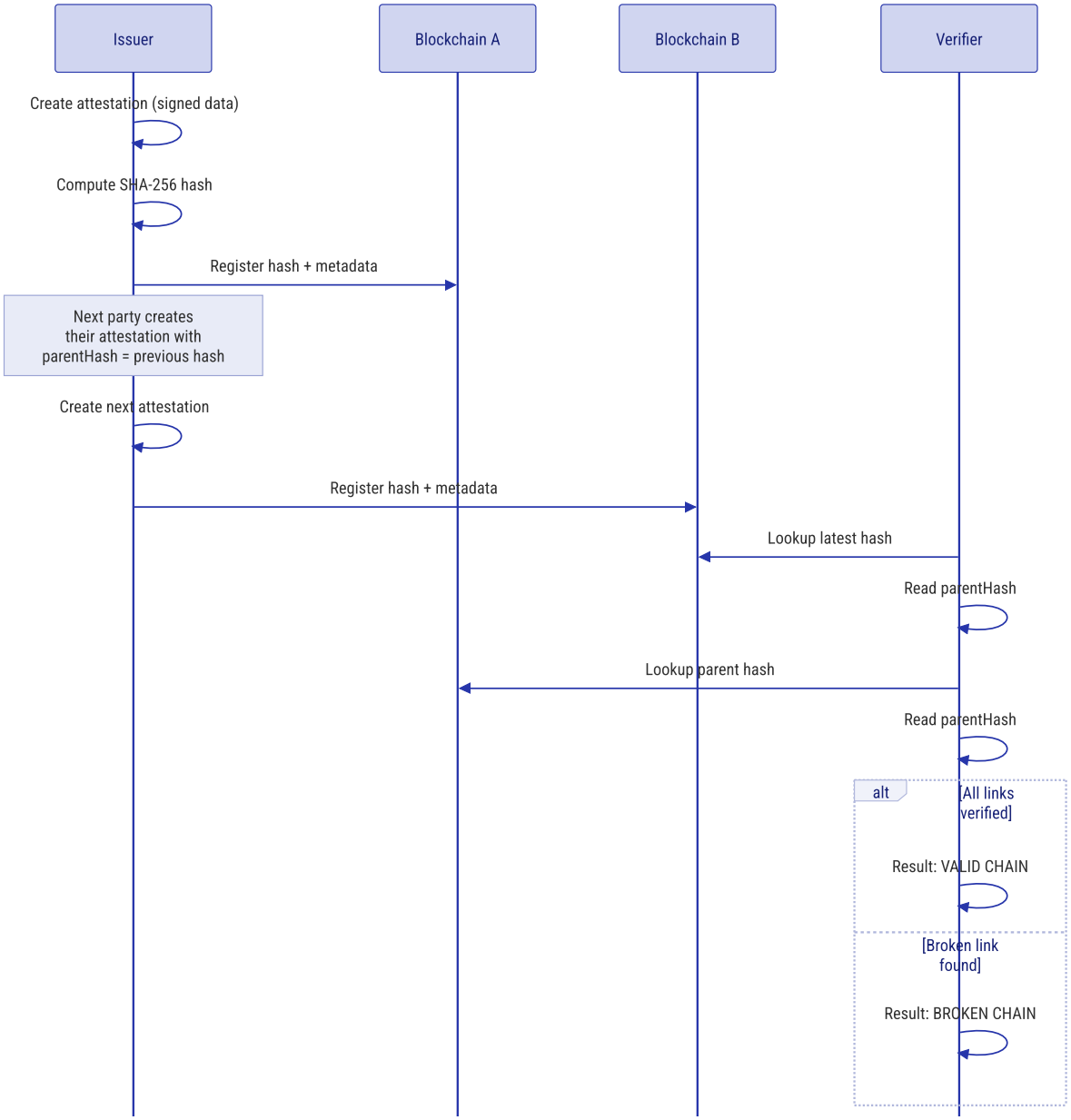
The DPP rollout follows a phased timeline. By July 2026, the EU will deploy a central DPP registry. Battery passports become mandatory from February 2027 for EV and industrial batteries. Textiles and iron/steel requirements are expected to follow in mid-2027. Electronics, furniture, and vehicles will be phased in between 2028 and 2029, with full adoption across remaining product categories by 2030.

The DPP mandates lifecycle tracking from production through end-of-life, covering material composition, carbon footprint, recycled content, durability metrics, and repair instructions. This is precisely the kind of structured, multi-party, lifecycle data that a provenance protocol is designed to capture.

The protocol presented in this paper is not a DPP implementation – the regulatory requirements are far more specific than what a general provenance protocol addresses. However, the cross-chain attestation model could serve as a technical building block for DPP systems, particularly for supply chains where different participants use different blockchain networks. The attestation chain model maps naturally to DPP lifecycle tracking: an origin attestation for manufacturing, transfer attestations for supply chain handoffs, authentication attestations for compliance checks, and event attestations for maintenance, repair, and end-of-life processing. This potential application is explored further in Section 8.

3. The Protocol

How It Works



The protocol defines four stages:

1. **Attestation creation.** A party creates a structured attestation about an asset – declaring its origin, recording a transfer, vouching for its authenticity, or documenting a significant event. The attestation includes a `parentHash` field that references the SHA-256 hash of the previous attestation in the chain (or zero for the origin attestation). The issuer signs the attestation using their cryptographic key.

2. **On-chain anchoring.** The issuer computes the SHA-256 hash of the complete attestation and registers this hash on a blockchain of their choice. The on-chain record stores the hash, the issuer's identifier, the parent hash, the attestation type, and a timestamp. Crucially, the parent hash reference is to the *attestation content*, not to a specific on-chain record. This is what enables cross-chain linking.

3. **Chain building.** Over time, successive parties add attestations to the chain, each referencing the previous one via its content hash. Different parties may anchor on different blockchains. The result is a linked chain of attestations where the links are content-based, not location-based.

4. **Cross-chain verification.** A verifier starts with the most recent attestation hash and walks the chain backward. For each hash, the verifier queries all known chains until it finds the record, reads the parent hash, and continues. If every link resolves to a registered, non-revoked attestation on some chain, and the chain terminates at an origin attestation (parent hash of zero), the provenance is verified. If any link fails to resolve, the chain is broken.

Attestation Types

The protocol defines four attestation types, chosen to cover the most common provenance events:

Type	Purpose	Key Fields	Parent Required
OriginAttestation	Establish asset identity	assetType, assetDescription, creationDate, creatorName, documentationHash	No (chain root)
TransferAttestation	Change of custody	fromDid, toDid, transferDate, transferType, supportingDocumentHash	Yes
AuthenticationAttestation	Expert vouches for asset	authenticatorDid, assessmentType, assessmentSummary, documentHash	Yes
EventAttestation	Significant event	eventType, eventDescription, eventDate, documentHash	Yes

OriginAttestation is the root of every provenance chain. It is created by the asset's originator – the artist who painted a painting, the manufacturer who built a watch, the winery that bottled a vintage. It establishes the asset's identity and initial documentation. Its `parentHash` is always zero (a 32-byte zero value), signaling the beginning of the chain.

TransferAttestation records a change of custody. The `transferType` field distinguishes between sales, gifts, consignments, and loans – an important distinction because consignment and loan do not transfer ownership. The `supportingDocumentHash` can reference an off-chain document such as a bill of sale.

AuthenticationAttestation records an expert or institution vouching for the asset. The `assessmentType` field covers authentication (confirming genuineness), appraisal (establishing value), condition reports, and restoration reports. Multiple independent authentication attestations strengthen provenance – a painting authenticated by three independent experts has stronger provenance than one authenticated by a single party.

EventAttestation covers significant events that do not change ownership: restoration work, exhibition history, damage, insurance claims, or conservation treatment. These events are important for provenance completeness – a painting's condition history affects its value and authenticity assessment.

Chain Linking

The chain linking mechanism is the protocol's core contribution. Each attestation (except the origin) includes a `parentHash` that references the SHA-256 hash of the previous attestation. This creates a linked list structure, but unlike a blockchain's block chain, the links are between content hashes, not between on-chain records.

This distinction is critical. If attestation A is anchored on Ethereum and attestation B (with `parentHash = hash(A)`) is anchored on IOTA, the link between them exists in the attestation content, not in any on-chain pointer. The Ethereum contract does not know about the IOTA contract, and vice versa. The verifier stitches the chain together by looking up each hash across all known chains.

This design means that adding a new blockchain to the ecosystem requires only deploying the provenance registry contract on that chain. No existing contracts need to be modified. No cross-chain messaging infrastructure is needed. The protocol is cross-chain by construction, not by integration.

Trust Model

The protocol uses an **open issuer model**: any party can create and anchor an attestation. There is no gatekeeper that decides who is allowed to attest. This mirrors how provenance works in practice – anyone can write a certificate of authenticity, but the certificate's value depends on the reputation of the person who wrote it.

Trust is the verifier's responsibility. A verifier evaluates provenance by considering not just the chain's integrity (are all links valid?) but also the identities and reputations of the attestors. An authentication attestation from a recognized expert carries more weight than one from an unknown party. A transfer attestation from a major auction house is more credible than one from an unverifiable source.

Revocation is access-controlled. Only the address that originally registered an attestation hash can revoke it. This ensures that attestors can withdraw their attestations (for example, if an authentication is later found to be incorrect) while preventing others from tampering with the chain.

Cross-Chain Verification

The verification algorithm walks the attestation chain from the most recent entry back to the origin:

```
function verifyChain(latestHash, knownChains):
    currentHash = latestHash
    chain = []

    while currentHash != ZERO_HASH:
        record = null
        for each chain in knownChains:
            record = chain.getAttestationStatus(currentHash)
```

```
            if record exists:
                break

        if record is null:
            return BROKEN_CHAIN (hash not found on any chain)

        if record.revoked:
            return REVOKED (attestation at currentHash was revoked)

        chain.prepend(record)
        currentHash = record.parentHash

    if chain[0].attestationType != Origin:
        return INVALID (chain does not start with origin)

    return VALID (complete chain from origin to latest)
```

The verification is linear in the number of attestations and, for each attestation, linear in the number of known chains. In practice, a lookup table mapping known hashes to their chain can reduce this to a single RPC call per attestation. The demo includes such a lookup table.

Advantages

Cross-chain by design. No cross-chain messaging, bridges, or relay networks are required. The protocol works across any set of blockchains that implement the registry contract.

Open and permissionless. No platform lock-in. Any party can deploy the registry on a new chain, create attestations, and participate in provenance chains. If a platform goes offline (as Everledger did), the on-chain records remain accessible.

Content-addressed linking. Attestation chains are linked by content hashes, not by chain-specific identifiers. This makes the chain portable and verifiable independently of where each attestation is anchored.

Cumulative trust. Provenance strength increases with each independent attestation. Multiple authentication attestations from different experts, transfer attestations from reputable institutions, and event attestations documenting condition history collectively build a provenance record that is difficult to fabricate.

Minimal on-chain footprint. Only hashes and metadata go on-chain. Attestation content, supporting documents, and images stay off-chain, keeping costs low and preserving privacy.

Limitations

No runtime attestation signing. The current proof of concept uses pre-computed attestation hashes rather than signing attestations at runtime. A full implementation would include JWT-based attestation signing using Ed25519 keys and `did:key` identifiers, but this is deferred to keep the PoC focused on the protocol mechanics and cross-chain verification.

Off-chain data availability. The on-chain hash proves that an attestation existed, but the attestation content must be stored and retrievable off-chain. If the off-chain storage is lost, the hash alone cannot reconstruct the attestation. A production system would need a content-addressed storage layer (such as IPFS or a dedicated attestation store).

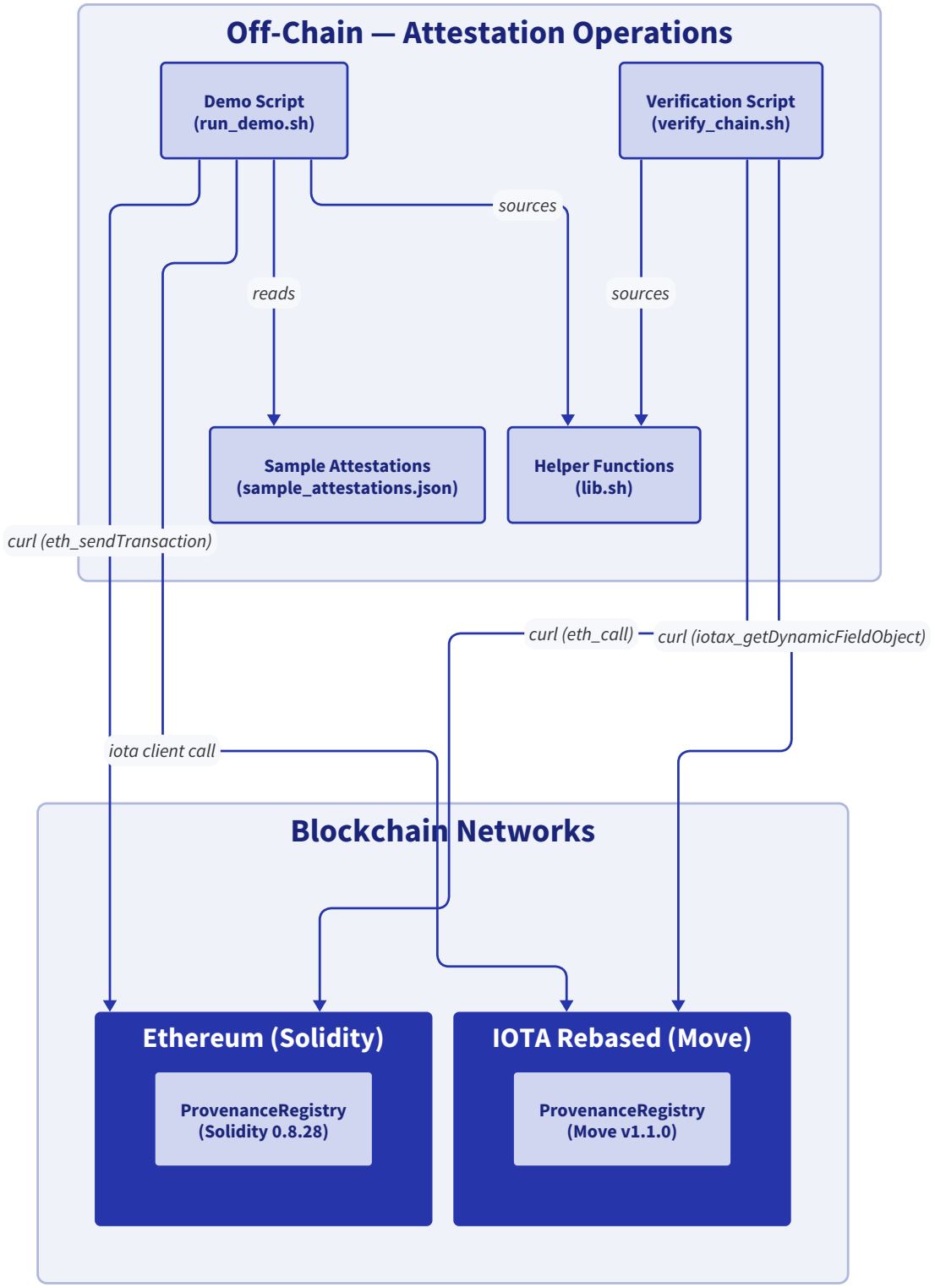
Trust bootstrapping. The protocol does not address how a verifier discovers which attestors to trust. In practice, this would involve attestor directories, institutional trust frameworks, or reputation systems.

No parent validation on-chain. The smart contract does not validate that a parent hash exists on-chain when registering a new attestation. This is a deliberate design choice – the parent may be on a different chain – but it means that invalid parent references are only detected during verification, not during registration.

Chain discovery. The verifier must know which chains to query. The protocol does not include a discovery mechanism for finding which chain holds a particular attestation. In the demo, a lookup table maps hashes to chains. A production system would need either a registry of registries or a broadcast mechanism.

4. Architecture

System Overview



The system has two layers. The **off-chain layer** consists of shell scripts that create attestations, interact with both blockchains, and perform cross-chain verification. The **blockchain layer** consists of two independent registry contracts – one on Ethereum, one on IOTA – that store attestation records. The contracts are identical in interface but implemented in their respective smart contract languages (Solidity and Move).

The key architectural principle is that the two chains know nothing about each other. There are no cross-chain messages, no bridges, and no shared state. The cross-chain linking exists purely in the attestation content (via parent hash references) and is resolved by the off-chain verification script.

Smart Contract Design

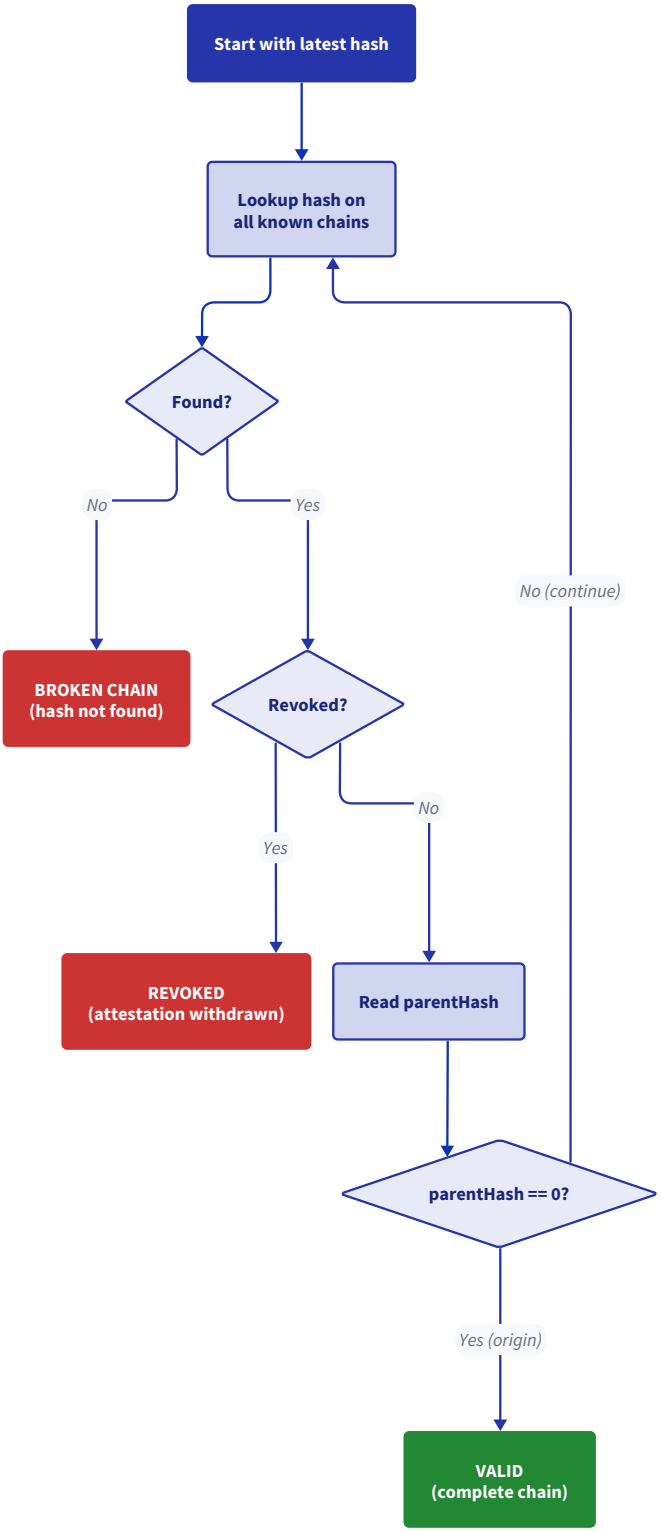
Both contracts implement the same logical interface:

- `registerAttestation(hash, issuerDid, parentHash, attestationType)` – record an attestation hash with metadata
- `revokeAttestation(hash)` – permanently revoke (original registrant only)
- `getAttestationStatus(hash)` – read the full on-chain record
- `isAttestationValid(hash)` – convenience check (registered and not revoked)
- `getParentHash(hash)` – return the parent hash for chain traversal

On EVM, the contract uses `mapping(bytes32 => AttestationRecord)` for storage, with a separate `mapping(bytes32 => address)` for registrant access control. On IOTA, the contract uses a shared `Registry` object containing a `Table<vector<u8>, AttestationRecord>`.

A critical design decision is that the contract does **not** validate that the parent hash exists on-chain when registering a new attestation. This is deliberate: the parent might be registered on a different blockchain. Validation of parent references happens during cross-chain verification, not during registration. This keeps the contract simple and chain-independent.

Cross-Chain Verification Flow



On-Chain vs Off-Chain Data

All bulk data stays off-chain. The blockchain stores only the minimum needed for verification:

On-Chain (per attestation)	Off-Chain
Attestation hash (32 bytes)	Full attestation content
Issuer DID (string)	Supporting documents
Parent hash (32 bytes)	Images, condition reports
Attestation type (enum)	Assessment summaries
Timestamp	Transfer records
Revoked flag	Provenance narratives
Registrant address	Signed JWT attestations

This keeps on-chain costs low while preserving the blockchain's core value: an immutable, tamper-proof audit trail that anyone can verify.

5. Smart Contract Implementation

ProvenanceRegistry.sol

The Solidity contract manages the attestation registry on EVM chains.

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.28;

/// @title ProvenanceRegistry
/// @notice On-chain registry for cross-chain asset provenance attestations.
/// Each attestation is identified by the SHA-256 hash of its off-chain content.
/// Parent hash references enable chain linking; parents may reside on other chains,
/// so the contract deliberately does NOT validate parent existence on-chain.
contract ProvenanceRegistry {

    // -- Attestation types ------------------------------------------------

    enum AttestationType {
        Origin,          // 0 - asset creation / first registration
        Transfer,        // 1 - change of custody
        Authentication,  // 2 - expert vouches for authenticity / condition
        Event            // 3 - restoration, exhibition, damage, insurance, etc.
    }

    // -- Data structures --------------------------------------------------

    struct AttestationRecord {
        string  issuerDid;        // DID of the party that created the attestation
        bytes32 parentHash;       // hash of the previous attestation (0x00..00 for origin)
        AttestationType aType;    // attestation category
        uint256 timestamp;        // block.timestamp at registration
        bool    revoked;          // true once revoked by the original registrant
        bool    exists;           // guard against default-value reads
    }

    // -- Storage ----------------------------------------------------------

    /// @dev attestation hash => record
    mapping(bytes32 => AttestationRecord) private records;

    /// @dev attestation hash => address that registered it (for revocation ACL)
    mapping(bytes32 => address) private registrants;

    // -- Events -----------------------------------------------------------

    event AttestationRegistered(
        bytes32 indexed hash,
        string  issuerDid,
        bytes32 parentHash,
        AttestationType aType,
        uint256 timestamp
    );
```

```
event AttestationRevoked(
    bytes32 indexed hash,
    uint256 timestamp
);

// -- Custom errors ----------------------------------------------------

error AttestationAlreadyRegistered(bytes32 hash);
error AttestationNotFound(bytes32 hash);
error NotOriginalRegistrant(bytes32 hash);
error AttestationAlreadyRevoked(bytes32 hash);
error EmptyIssuer();

// -- Public functions -------------------------------------------------

/// @notice Register an attestation hash with its metadata.
/// @param hash         SHA-256 hash of the full off-chain attestation
/// @param issuerDid    DID of the issuer (e.g. did:key:z6Mk...)
/// @param parentHash   hash of the parent attestation (zero for origin)
/// @param aType        attestation category
function registerAttestation(
    bytes32 hash,
    string calldata issuerDid,
    bytes32 parentHash,
    AttestationType aType
) external {
    if (bytes(issuerDid).length == 0) revert EmptyIssuer();
    if (records[hash].exists) revert AttestationAlreadyRegistered(hash);

    records[hash] = AttestationRecord({
        issuerDid:  issuerDid,
        parentHash: parentHash,
        aType:      aType,
        timestamp:  block.timestamp,
        revoked:    false,
        exists:     true
    });

    registrants[hash] = msg.sender;

    emit AttestationRegistered(hash, issuerDid, parentHash, aType, block.timestamp);
}

/// @notice Revoke an attestation. Only the original registrant may call this.
/// @param hash  the attestation hash to revoke
function revokeAttestation(bytes32 hash) external {
    if (!records[hash].exists)       revert AttestationNotFound(hash);
    if (records[hash].revoked)       revert AttestationAlreadyRevoked(hash);
    if (registrants[hash] != msg.sender) revert NotOriginalRegistrant(hash);

    records[hash].revoked = true;

    emit AttestationRevoked(hash, block.timestamp);
}
```

```
    /// @notice Read the full on-chain record for an attestation.
    /// @param hash  the attestation hash to query
    /// @return issuerDid   DID of the issuer
    /// @return parentHash  hash of the parent attestation
    /// @return aType       attestation category
    /// @return timestamp   block.timestamp at registration
    /// @return revoked     whether the attestation has been revoked
    /// @return exists      whether the attestation exists at all
    function getAttestationStatus(bytes32 hash)
        external
        view
        returns (
            string  memory issuerDid,
            bytes32 parentHash,
            AttestationType aType,
            uint256 timestamp,
            bool    revoked,
            bool    exists
        )
    {
        AttestationRecord storage r = records[hash];
        return (r.issuerDid, r.parentHash, r.aType, r.timestamp, r.revoked, r.exists);
    }

    /// @notice Convenience check: registered and not revoked.
    /// @param hash  the attestation hash to check
    /// @return true if the attestation is registered and has not been revoked
    function isAttestationValid(bytes32 hash) external view returns (bool) {
        AttestationRecord storage r = records[hash];
        return r.exists && !r.revoked;
    }

    /// @notice Return the parent hash for chain traversal.
    /// @param hash  the attestation hash to query
    /// @return the parent hash (zero for origin attestations)
    function getParentHash(bytes32 hash) external view returns (bytes32) {
        if (!records[hash].exists) revert AttestationNotFound(hash);
        return records[hash].parentHash;
    }
}
```

The contract uses established Solidity patterns: custom errors for gas efficiency, events for off-chain indexing, and a separate registrant mapping for revocation access control. The `exists` flag prevents default-value reads from appearing valid.

provenance_registry.move

The Move contract implements the same interface for IOTA Rebased.

```
/// Module: provenance_registry
/// On-chain registry for cross-chain asset provenance attestations on IOTA.
/// Mirrors the Solidity ProvenanceRegistry interface: register, revoke,
/// query, and chain-walk via parent hash references.
#[allow(unused_const)]
```

```move
module provenance_registry::provenance_registry {

    use iota::table::{Self, Table};
    use iota::clock::{Self, Clock};

    // -- Error codes ----------------------------------------------------------

    const E_ATTESTATION_ALREADY_REGISTERED: u64 = 1;
    const E_ATTESTATION_NOT_FOUND: u64         = 2;
    const E_NOT_ORIGINAL_REGISTRANT: u64       = 3;
    const E_ATTESTATION_ALREADY_REVOKED: u64   = 4;
    const E_EMPTY_ISSUER: u64                  = 5;

    // -- Attestation type constants -------------------------------------------

    const TYPE_ORIGIN: u8         = 0;
    const TYPE_TRANSFER: u8       = 1;
    const TYPE_AUTHENTICATION: u8 = 2;
    const TYPE_EVENT: u8          = 3;

    // -- Data structures ------------------------------------------------------

    /// Record stored per attestation hash in the shared Table.
    public struct AttestationRecord has store, drop, copy {
        issuer_did:    vector<u8>,   // DID of the attestation issuer
        parent_hash:   vector<u8>,   // hash of parent attestation (empty for origin)
        atype:         u8,           // attestation type constant
        timestamp_ms:  u64,          // milliseconds from Clock
        revoked:       bool,         // true once revoked by original registrant
        registrant:    address,      // address that registered (for revocation ACL)
    }

    /// Shared registry object containing all attestation records.
    public struct Registry has key {
        id: UID,
        attestations: Table<vector<u8>, AttestationRecord>,
    }

    // -- Initialization -------------------------------------------------------

    /// Create the shared Registry on module publish.
    fun init(ctx: &mut TxContext) {
        let registry = Registry {
            id: object::new(ctx),
            attestations: table::new(ctx),
        };
        transfer::share_object(registry);
    }

    // -- Public entry functions -----------------------------------------------

    /// Register an attestation hash with metadata.
    /// Parent hash is NOT validated on-chain (it may be on another chain).
    public entry fun register_attestation(
        registry: &mut Registry,
        hash: vector<u8>,
```

```
        issuer_did: vector<u8>,
        parent_hash: vector<u8>,
        attestation_type: u8,
        clock: &Clock,
        ctx: &mut TxContext,
    ) {
        assert!(!std::vector::is_empty(&issuer_did), E_EMPTY_ISSUER);
        assert!(
            !table::contains(&registry.attestations, hash),
            E_ATTESTATION_ALREADY_REGISTERED,
        );

        let record = AttestationRecord {
            issuer_did,
            parent_hash,
            atype: attestation_type,
            timestamp_ms: clock::timestamp_ms(clock),
            revoked: false,
            registrant: tx_context::sender(ctx),
        };

        table::add(&mut registry.attestations, hash, record);
    }

    /// Revoke an attestation. Only the original registrant may call.
    public entry fun revoke_attestation(
        registry: &mut Registry,
        hash: vector<u8>,
        ctx: &mut TxContext,
    ) {
        assert!(
            table::contains(&registry.attestations, hash),
            E_ATTESTATION_NOT_FOUND,
        );

        let record = table::borrow_mut(&mut registry.attestations, hash);

        assert!(!record.revoked, E_ATTESTATION_ALREADY_REVOKED);
        assert!(
            record.registrant == tx_context::sender(ctx),
            E_NOT_ORIGINAL_REGISTRANT,
        );

        record.revoked = true;
    }

    /// Read the full on-chain record for an attestation.
    /// In the demo, reads use iotax_getDynamicFieldObject rather than
    /// calling this function, because devInspect proved problematic on
    /// IOTA v1.1.0 and direct dynamic field reads are a more natural
    /// approach for IOTA's object model.
    public fun get_attestation_status(
        registry: &Registry,
        hash: vector<u8>,
    ): (vector<u8>, vector<u8>, u8, u64, bool) {
        assert!(
```

```
            table::contains(&registry.attestations, hash),
            E_ATTESTATION_NOT_FOUND,
        );

        let record = table::borrow(&registry.attestations, hash);
        (
            record.issuer_did,
            record.parent_hash,
            record.atype,
            record.timestamp_ms,
            record.revoked,
        )
    }

    /// Convenience check: registered and not revoked.
    public fun is_attestation_valid(
        registry: &Registry,
        hash: vector<u8>,
    ): bool {
        if (!table::contains(&registry.attestations, hash)) {
            return false
        };
        let record = table::borrow(&registry.attestations, hash);
        !record.revoked
    }

    /// Return the parent hash for chain traversal.
    public fun get_parent_hash(
        registry: &Registry,
        hash: vector<u8>,
    ): vector<u8> {
        assert!(
            table::contains(&registry.attestations, hash),
            E_ATTESTATION_NOT_FOUND,
        );

        let record = table::borrow(&registry.attestations, hash);
        record.parent_hash
    }

    // -- Tests ---------------------------------------------------------------

    #[test_only]
    use iota::test_scenario;
    #[test_only]
    use iota::clock as test_clock;

    #[test]
    /// Register an origin attestation and verify it can be queried.
    fun test_register_and_query() {
        let owner = @0xA;
        let mut scenario = test_scenario::begin(owner);

        // Publish module, creating the shared Registry
        {
            init(test_scenario::ctx(&mut scenario));
```

```
        };

        test_scenario::next_tx(&mut scenario, owner);

        {
            let mut registry = test_scenario::take_shared<Registry>(&scenario);
            let mut clock = test_clock::create_for_testing(test_scenario::ctx(&mut scenario));
            test_clock::set_for_testing(&mut clock, 1000);

            let hash = b"attestation_hash_01";
            let issuer = b"did:key:z6MkArtist";
            let parent = b"";

            register_attestation(
                &mut registry,
                hash,
                issuer,
                parent,
                TYPE_ORIGIN,
                &clock,
                test_scenario::ctx(&mut scenario),
            );

            let (did, ph, at, ts, rev) = get_attestation_status(&registry, hash);
            assert!(did == b"did:key:z6MkArtist");
            assert!(ph == b"");
            assert!(at == TYPE_ORIGIN);
            assert!(ts == 1000);
            assert!(!rev);
            assert!(is_attestation_valid(&registry, hash));

            test_clock::destroy_for_testing(clock);
            test_scenario::return_shared(registry);
        };

        test_scenario::end(scenario);
    }

    #[test]
    /// Register and then revoke an attestation.
    fun test_revoke() {
        let owner = @0xA;
        let mut scenario = test_scenario::begin(owner);

        {
            init(test_scenario::ctx(&mut scenario));
        };

        test_scenario::next_tx(&mut scenario, owner);

        {
            let mut registry = test_scenario::take_shared<Registry>(&scenario);
            let mut clock = test_clock::create_for_testing(test_scenario::ctx(&mut scenario));
            test_clock::set_for_testing(&mut clock, 2000);

            let hash = b"attestation_hash_02";
```

```
            register_attestation(
                &mut registry, hash, b"did:key:z6MkGallery", b"parent01",
                TYPE_TRANSFER, &clock, test_scenario::ctx(&mut scenario),
            );

            assert!(is_attestation_valid(&registry, hash));

            revoke_attestation(
                &mut registry, hash, test_scenario::ctx(&mut scenario),
            );

            assert!(!is_attestation_valid(&registry, hash));
            let (_, _, _, _, rev) = get_attestation_status(&registry, hash);
            assert!(rev);

            test_clock::destroy_for_testing(clock);
            test_scenario::return_shared(registry);
        };

        test_scenario::end(scenario);
    }

    #[test]
    /// Verify parent hash is stored correctly for chain linking.
    fun test_chain_linking() {
        let owner = @0xA;
        let mut scenario = test_scenario::begin(owner);

        {
            init(test_scenario::ctx(&mut scenario));
        };

        test_scenario::next_tx(&mut scenario, owner);

        {
            let mut registry = test_scenario::take_shared<Registry>(&scenario);
            let mut clock = test_clock::create_for_testing(test_scenario::ctx(&mut scenario));
            test_clock::set_for_testing(&mut clock, 3000);

            let origin_hash = b"origin_hash";
            let child_hash = b"child_hash";

            register_attestation(
                &mut registry, origin_hash, b"did:key:z6MkArtist", b"",
                TYPE_ORIGIN, &clock, test_scenario::ctx(&mut scenario),
            );

            register_attestation(
                &mut registry, child_hash, b"did:key:z6MkGallery", origin_hash,
                TYPE_TRANSFER, &clock, test_scenario::ctx(&mut scenario),
            );

            let parent = get_parent_hash(&registry, child_hash);
            assert!(parent == origin_hash);

            let origin_parent = get_parent_hash(&registry, origin_hash);
```

```
            assert!(origin_parent == b"");

            test_clock::destroy_for_testing(clock);
            test_scenario::return_shared(registry);
        };

        test_scenario::end(scenario);
    }

    #[test]
    #[expected_failure(abort_code = E_ATTESTATION_ALREADY_REGISTERED)]
    /// Attempting to register the same hash twice should fail.
    fun test_duplicate_prevention() {
        let owner = @0xA;
        let mut scenario = test_scenario::begin(owner);

        {
            init(test_scenario::ctx(&mut scenario));
        };

        test_scenario::next_tx(&mut scenario, owner);

        {
            let mut registry = test_scenario::take_shared<Registry>(&scenario);
            let mut clock = test_clock::create_for_testing(test_scenario::ctx(&mut scenario));
            test_clock::set_for_testing(&mut clock, 4000);

            let hash = b"duplicate_hash";
            register_attestation(
                &mut registry, hash, b"did:key:z6MkTest", b"",
                TYPE_ORIGIN, &clock, test_scenario::ctx(&mut scenario),
            );

            // Second registration with same hash should abort
            register_attestation(
                &mut registry, hash, b"did:key:z6MkOther", b"",
                TYPE_ORIGIN, &clock, test_scenario::ctx(&mut scenario),
            );

            test_clock::destroy_for_testing(clock);
            test_scenario::return_shared(registry);
        };

        test_scenario::end(scenario);
    }

    #[test]
    #[expected_failure(abort_code = E_NOT_ORIGINAL_REGISTRANT)]
    /// Only the original registrant can revoke an attestation.
    fun test_non_registrant_revocation() {
        let owner = @0xA;
        let other = @0xB;
        let mut scenario = test_scenario::begin(owner);

        {
            init(test_scenario::ctx(&mut scenario));
```

```
        };

        // Owner registers
        test_scenario::next_tx(&mut scenario, owner);
        {
            let mut registry = test_scenario::take_shared<Registry>(&scenario);
            let mut clock = test_clock::create_for_testing(test_scenario::ctx(&mut scenario));
            test_clock::set_for_testing(&mut clock, 5000);

            register_attestation(
                &mut registry, b"acl_hash", b"did:key:z6MkOwner", b"",
                TYPE_ORIGIN, &clock, test_scenario::ctx(&mut scenario),
            );

            test_clock::destroy_for_testing(clock);
            test_scenario::return_shared(registry);
        };

        // Other tries to revoke -- should fail
        test_scenario::next_tx(&mut scenario, other);
        {
            let mut registry = test_scenario::take_shared<Registry>(&scenario);

            revoke_attestation(
                &mut registry, b"acl_hash", test_scenario::ctx(&mut scenario),
            );

            test_scenario::return_shared(registry);
        };

        test_scenario::end(scenario);
    }

    #[test]
    #[expected_failure(abort_code = E_ATTESTATION_NOT_FOUND)]
    /// Querying a nonexistent attestation should fail.
    fun test_nonexistent_attestation() {
        let owner = @0xA;
        let mut scenario = test_scenario::begin(owner);

        {
            init(test_scenario::ctx(&mut scenario));
        };

        test_scenario::next_tx(&mut scenario, owner);

        {
            let registry = test_scenario::take_shared<Registry>(&scenario);

            // This should abort -- hash was never registered
            let (_did, _ph, _at, _ts, _rev) =
                get_attestation_status(&registry, b"nonexistent_hash");

            test_scenario::return_shared(registry);
        };
```

```
        test_scenario::end(scenario);
    }
}
```

Move.toml for the IOTA contract:

```
[package]
name = "provenance_registry"
edition = "2024.beta"

[dependencies]
Iota = { git = "https://github.com/iotaledger/iota.git", subdir = "crates/iota-framework/packages/io

[addresses]
provenance_registry = "0x0"
```

The Move contract uses a shared `Registry` object with `Table`, created via `transfer::share_object` in `init`. Timestamps come from IOTA's `Clock` object. Error codes are module-level constants, and test cases use `expected_failure` annotations with numeric abort codes. Table keys are passed by value per the v1.1.0 Table API.

6. Demonstration

Prerequisites

- Docker and Docker Compose
- The Ethereum Local Testing Starter Pack
- The IOTA Local Testing Starter Pack

Both starter packs must be running simultaneously. The demo uses the Ethereum starter pack's Hardhat node (port 41545) and the IOTA starter pack's full node (port 43900).

Setup

Start both networks:

```
# Terminal 1: Ethereum
cd /path/to/Ethereum-Local-Testing-Starter-Pack
./eth.sh start

# Terminal 2: IOTA
cd /path/to/IOTA-Local-Testing-Starter-Pack
./iota.sh start
```

Configure the environment:

```
cd cross-chain-asset-provenance
cp .env.example .env
# Edit .env to set ETH_STARTER_PACK_DIR and IOTA_STARTER_PACK_DIR
```

Run the complete demonstration:

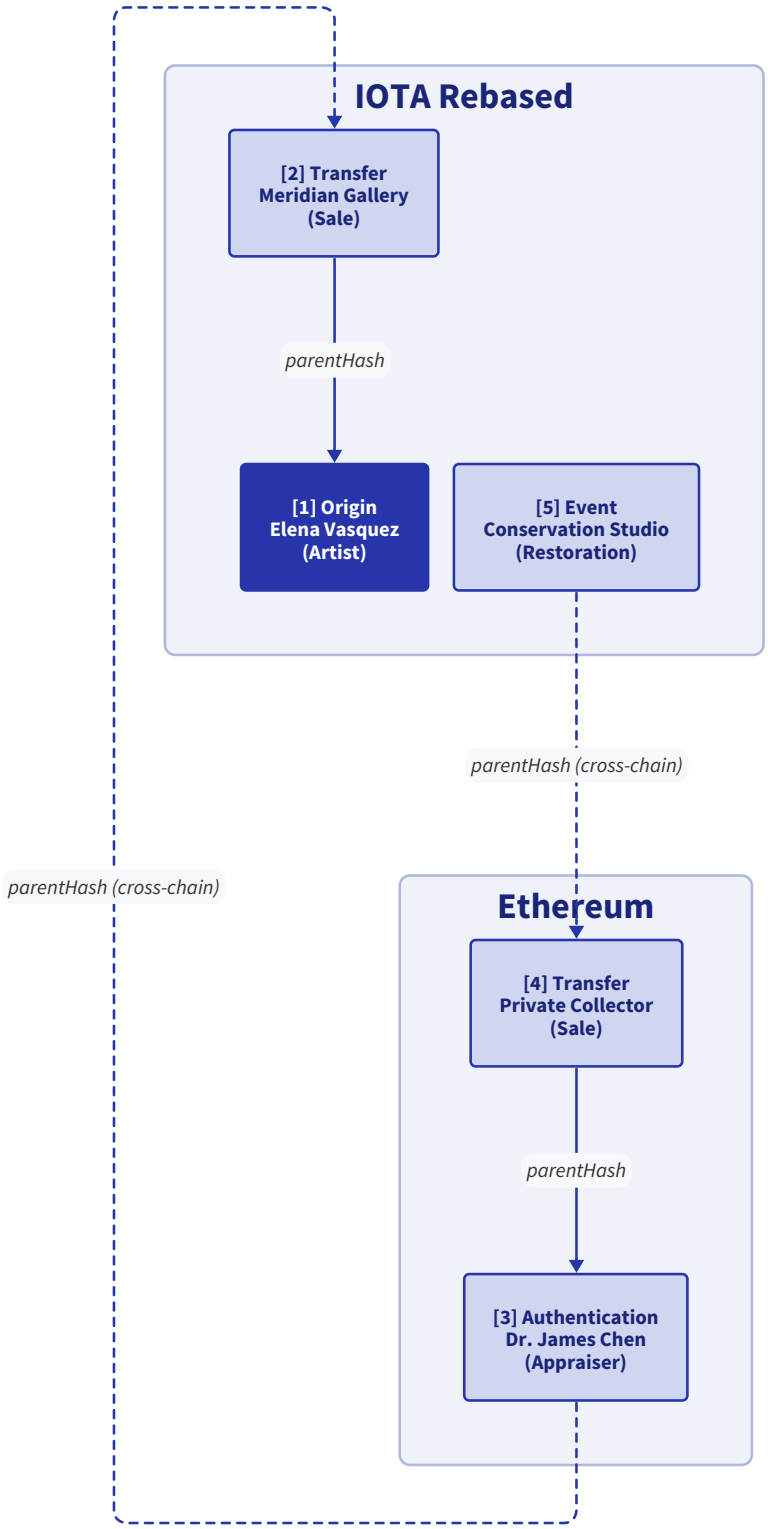
```
./run_demo.sh
```

The Painting Scenario

The demo simulates a painting's journey through five parties, with attestations alternating between the two chains:

Step	Attestation	Party	Chain	Type
1	Origin	Artist (Elena Vasquez)	IOTA	OriginAttestation
2	Gallery acquisition	Meridian Gallery	IOTA	TransferAttestation
3	Authentication	Dr. James Chen (appraiser)	Ethereum	AuthenticationAttestation
4	Collector purchase	Private collector	Ethereum	TransferAttestation
5	Restoration	Conservation Studio	IOTA	EventAttestation

The following diagram shows the provenance chain for the painting, with attestations grouped by the chain they are anchored on. The arrows represent parent hash links – note how they cross between chains.



This sequence is designed to exercise cross-chain verification at every step. The verification script must cross from IOTA to Ethereum (at step 3) and back to IOTA (at step 2) to walk the complete chain.

The `run_demo.sh` script performs the following operations:

1. Compiles the Move contract via the starter pack's `iota-tools` container and deploys it on IOTA using `iota client publish`
2. Deploys the Solidity contract on Ethereum using `curl` against the Hardhat JSON-RPC with pre-compiled bytecode via `eth_sendTransaction`
3. Registers attestations 1 and 2 on IOTA using `docker compose run --rm iota-tools iota client call ...`
4. Registers attestations 3 and 4 on Ethereum using `curl` with `eth_sendTransaction` and ABI-encoded calldata
5. Registers attestation 5 on IOTA
6. Reads back each attestation's on-chain status via the appropriate chain's RPC
7. Runs the cross-chain verification walk from attestation 5 back to attestation 1
8. Demonstrates forgery detection: attempts to verify a hash that references a non-existent parent

For Ethereum operations, the demo uses Hardhat's unlocked accounts (which support `eth_sendTransaction` without client-side signing). Function selectors and arguments are ABI-encoded as zero-padded hex strings. For example, the `registerAttestation` function selector is the first 4 bytes of the Keccak-256 hash of its signature, and the `bytes32` hash argument is zero-padded to 32 bytes.

For IOTA operations, write operations use the `iota` CLI binary inside the starter pack's `iota-tools` container, which handles transaction signing internally. Read operations use `curl` against the IOTA JSON-RPC endpoint's `iotax_getDynamicFieldObject` method, which reads attestation records directly from the Registry's Table without requiring a transaction.

Forgery Detection Scenario

After the legitimate chain is built, the demo demonstrates forgery detection:

1. An attacker creates a fake transfer attestation claiming to have acquired the painting
2. The fake attestation references a parent hash that does not exist on any chain
3. The verification script detects the broken link and reports the chain as invalid

This shows that the protocol's chain linking provides forgery detection at the structural level: any attestation that cannot be traced back to a registered origin through an unbroken chain of valid parent references is identified as suspicious.

Verification Output

The `verify_chain.sh` script produces output in the following format for each attestation in the chain:

```
======================================================================
  Cross-Chain Provenance Verification
======================================================================

Walking chain from: 0x5a1b3f62...4b01b4

  [1] Hash:   0x5a1b3f62...4b01b4
      Chain:  IOTA
      Type:   Event
      Issuer: did:key:z6MkConservationStudio
      Parent: 0x2e767ef5...6dbd4d
      Status: REGISTERED (valid)
```

```
  [2] Hash:   0x2e767ef5...6dbd4d
      Chain:  Ethereum
      Type:   Transfer
      Issuer: did:key:z6MkPrivateCollector
      Parent: 0x613700e7...f05fc0
      Status: REGISTERED (valid)

  [3] Hash:   0x613700e7...f05fc0
      Chain:  Ethereum
      Type:   Authentication
      Issuer: did:key:z6MkDrJamesChen
      Parent: 0x5911f4d6...0e77e3
      Status: REGISTERED (valid)

  [4] Hash:   0x5911f4d6...0e77e3
      Chain:  IOTA
      Type:   Transfer
      Issuer: did:key:z6MkMeridianGallery
      Parent: 0xb48e3b1f...d02086
      Status: REGISTERED (valid)

  [5] Hash:   0xb48e3b1f...d02086
      Chain:  IOTA
      Type:   Origin
      Issuer: did:key:z6MkArtistElenaVasquez
      Parent: 0x00000000...000000
      Status: REGISTERED (valid)

======================================================================
  Result: VALID PROVENANCE CHAIN
  Links verified: 5
  Chains traversed: IOTA, Ethereum
======================================================================
```

For the forgery scenario:

```
======================================================================
  Cross-Chain Forgery Detection
======================================================================

Walking chain from: 0x324a6abb...2a4bbd

  [1] Hash:   0x324a6abb...2a4bbd
      Chain:  Ethereum
      Type:   Transfer
      Issuer: did:key:z6MkForger
      Parent: 0xbad00000...000bad
      Status: REGISTERED (valid)

  [!] Hash:   0xbad0000000000000000000000000000000000000000000000000000000000bad
      Ethereum: NOT FOUND
      IOTA:     NOT FOUND

======================================================================
```

```
  Result: BROKEN CHAIN
  Reason: Parent hash 0xbad00000...000bad not registered on
          any known chain.
==========================================================================
```

7. Results

Summary

Metric	Ethereum	IOTA
Attestations registered	2 (+1 forged)	3
Registration cost	~162,000 gas per attestation	~3,900,000 NANOS per attestation
Revocation cost	~32,000 gas	~1,000,000 NANOS
Verification cost	0 (view call)	0 (direct read)
Contract language	Solidity 0.8.28	Move (IOTA v1.1.0)
Read mechanism	`eth_call` with ABI encoding	`iotax_getDynamicFieldObject`

Cross-Chain Verification Output

The painting scenario verification walks all five attestations across both chains:

Step	Type	Chain	Issuer	Status
1	Origin	IOTA	Artist (Elena Vasquez)	Valid
2	Transfer	IOTA	Meridian Gallery	Valid
3	Authentication	Ethereum	Dr. James Chen	Valid
4	Transfer	Ethereum	Private collector	Valid
5	Event	IOTA	Conservation Studio	Valid

All five links verified successfully. The verification script crossed from IOTA to Ethereum twice and from Ethereum to IOTA once during the chain walk.

The forgery detection scenario correctly identified the broken chain: the forged attestation was registered on Ethereum (any address can register), but its parent hash did not resolve on any known chain.

Observations

Cross-chain verification works without cross-chain infrastructure. The most significant result is that no bridges, relays, or cross-chain messaging was needed. The verification script simply queries each chain's RPC endpoint for each hash. The chains are completely independent – the cross-chain linking exists entirely in the attestation content.

The parent hash model provides structural forgery detection. A forger can register an attestation on any chain (the open issuer model allows this), but they cannot create a valid parent reference unless they know the hash of a legitimately registered attestation. If they reference a non-existent parent, the chain walk detects it. If they

reference a real attestation, they create a fork in the provenance tree, which a verifier can detect by checking whether multiple attestations claim the same parent.

Contract size and complexity are minimal. Both contracts are under 150 lines of meaningful code (excluding tests). The on-chain logic is deliberately simple: store a record, revoke it, read it. All complex logic (chain walking, cross-chain lookup, trust evaluation) lives off-chain. This is consistent with the philosophy that blockchain should be an audit and accountability layer, not an application platform.

Shell scripts are sufficient for a protocol demonstration. The entire demo runs without Python, without a CLI framework, and without any JWT signing libraries. This proves that the protocol's on-chain operations are simple RPC calls that can be invoked from any environment. A full PoC would add attestation signing, off-chain storage, and a user interface, but the protocol mechanics are demonstrable with minimal tooling.

Registration costs are modest. The EVM registration cost of approximately 162,000 gas is typical for a function that writes a struct with several fields to storage. IOTA costs are approximately 3,900,000 NANOS per registration, in a comparable range relative to its fee model. Verification remains free on both chains – EVM view calls cost zero gas, and IOTA dynamic field reads require no transaction.

8. Future Directions

Full PoC with JWT Signing

The most immediate next step is adding runtime attestation signing. Each attestation would be created as a JWT (Ed25519, `did:key`) containing the attestation fields, and the SHA-256 hash of the complete signed JWT would be anchored on-chain. The on-chain contracts would require no changes – they already store arbitrary 32-byte hashes. The signed JWTs would be stored off-chain (in a content-addressed store or a simple file system) and presented to verifiers alongside the on-chain record.

NFT Integration Layer

An optional extension could mint an NFT that references a provenance chain. The NFT would not replace the attestation chain (which remains the authoritative record) but would provide a convenient handle for marketplaces, wallets, and display applications. The NFT's metadata would include the latest attestation hash, the asset description, and a URI for retrieving the full provenance chain. This bridges the protocol with the existing NFT ecosystem without compromising the attestation-based trust model.

IoT Condition Monitoring

For assets where condition is critical (fine art, wine, sensitive instruments), IoT sensors could generate automated EventAttestations recording environmental conditions: temperature, humidity, vibration, light exposure. These attestations would be anchored at regular intervals, creating a continuous condition record that complements the human-generated provenance chain. The low cost of attestation registration (especially on IOTA, where mainnet transactions are feeless) makes high-frequency condition monitoring economically feasible.

Professional Credential Verification for Authenticators

Authentication attestations are only as trustworthy as the authenticator. A natural extension is to cross-reference authenticator identities against a verifiable credential registry – a system where professional qualifications are themselves cryptographically signed and blockchain-anchored. An appraiser's authentication attestation could be verified not just for its on-chain status but also against a verifiable credential confirming the appraiser's professional qualifications. This creates a nested trust model: the provenance protocol trusts the authenticator, and the credential protocol provides evidence for that trust.

EU Digital Product Passport Compliance

The attestation chain model maps naturally to the DPP lifecycle tracking requirements. An OriginAttestation captures manufacturing provenance. TransferAttestations record supply chain handoffs. AuthenticationAttestations document compliance checks and certifications. EventAttestations cover maintenance, repair, and end-of-life processing. A DPP adapter layer could translate between the protocol's attestation format and the specific data schemas required by ESPR delegated acts. The cross-chain capability is particularly relevant for DPP, since supply chain participants across different jurisdictions may use different blockchain networks.

Selective Disclosure of Provenance

Not all provenance information should be publicly visible. A collector may want to prove that a painting has an unbroken chain of custody without revealing the purchase price or the seller's identity. Selective disclosure techniques (such as SD-JWT or hash-based field commitments) could allow attestation holders to reveal specific fields while keeping others private. The on-chain hash would still verify the complete attestation's integrity, but the verifier would see only the disclosed fields.

Appendix A: Project File Reference

Repository Structure

```
cross-chain-asset-provenance/
|-- README.md
|-- compose.yml                    # For Move contract compilation (references starter pack)
|-- .env.example                   # Environment configuration template
|-- run_demo.sh                    # End-to-end demo script (both chains)
|-- contracts/
|   |-- evm/
|   |   `-- ProvenanceRegistry.sol # Solidity contract (0.8.28)
|   `-- iota/
|       |-- Move.toml              # Move package manifest (v1.1.0)
|       |-- sources/
|       |   `-- provenance_registry.move
|       `-- tests/
|           `-- provenance_registry_tests.move
|-- demo/
|   |-- verify_chain.sh            # Cross-chain provenance verification
|   |-- sample_attestations.json   # Pre-computed attestation data
|   `-- lib.sh                     # Shared helper functions (hex encoding, RPC calls)
`-- docs/
    `-- paper.md
```

Environment Variables

Variable	Required	Description
`EVM_RPC_URL`	Yes	Ethereum JSON-RPC endpoint
`EVM_ACCOUNT`	Yes	Unlocked Hardhat account address
`ETH_STARTER_PACK_DIR`	Yes	Path to the Ethereum Local Testing Starter Pack
`IOTA_RPC_URL`	Yes	IOTA full node JSON-RPC endpoint
`IOTA_STARTER_PACK_DIR`	Yes	Path to the IOTA Local Testing Starter Pack

Appendix B: Full Demonstration Output

B.1: Painting Scenario

```
======================================================================
  Step 0: Deploying Contracts
======================================================================

-> Building provenance_registry Move package via starter pack iota-tools...
FETCHING GIT DEPENDENCY https://github.com/iotaledger/iota.git
INCLUDING DEPENDENCY Iota
INCLUDING DEPENDENCY MoveStdlib
BUILDING provenance_registry

-> Deploying ProvenanceRegistry on IOTA...
   Package ID:  0xd528...e81d
   Registry ID: 0xdb08...8366

-> Compiling ProvenanceRegistry.sol via starter pack hardhat container...
-> Deploying ProvenanceRegistry on Ethereum...
   Contract: 0xe0cf...7c31

======================================================================
  Step 1: Origin Attestation (IOTA)
======================================================================

-> Registering: Artist creates painting
   Asset:   "Convergence at Dusk" by Elena Vasquez, oil on canvas, 2024
   Type:    Origin
   Issuer:  did:key:z6MkArtistElenaVasquez
   Parent:  0x0000000000000000000000000000000000000000000000000000000000000000
   Chain:   IOTA

Registered. Cost (NANOS): 3941200

======================================================================
  Step 2: Gallery Acquisition (IOTA)
======================================================================

-> Registering: Meridian Gallery acquires from artist
   Type:    Transfer (sale)
   Issuer:  did:key:z6MkMeridianGallery
   Parent:  0xb48e3b1f...d02086
   Chain:   IOTA

Registered. Cost (NANOS): 3918400

======================================================================
  Step 3: Authentication (Ethereum)
======================================================================

-> Registering: Dr. James Chen authenticates the painting
   Type:    Authentication
```

```
   Issuer:  did:key:z6MkDrJamesChen
   Parent:  0x5911f4d6...0e77e3
   Chain:   Ethereum

Registered. Gas used: 161936

========================================================================
  Step 4: Collector Purchase (Ethereum)
========================================================================

-> Registering: Private collector purchases from gallery
   Type:    Transfer (sale)
   Issuer:  did:key:z6MkPrivateCollector
   Parent:  0x613700e7...f05fc0
   Chain:   Ethereum

Registered. Gas used: 161996

========================================================================
  Step 5: Restoration (IOTA)
========================================================================

-> Registering: Conservation Studio performs restoration
   Type:    Event (restoration)
   Issuer:  did:key:z6MkConservationStudio
   Parent:  0x2e767ef5...6dbd4d
   Chain:   IOTA

Registered. Cost (NANOS): 3941200

========================================================================
  Cross-Chain Provenance Verification
========================================================================

Walking chain from: 0x5a1b3f62b6dc8ac32ee54e68e2131920df051c772b74ade221c4b744834b01b4

  [1] Hash:   0x5a1b3f62...4b01b4
      Chain:  IOTA
      Type:   Event
      Issuer: did:key:z6MkConservationStudio
      Parent: 0x2e767ef5...6dbd4d
      Status: REGISTERED (valid)

  [2] Hash:   0x2e767ef5...6dbd4d
      Chain:  Ethereum
      Type:   Transfer
      Issuer: did:key:z6MkPrivateCollector
      Parent: 0x613700e7...f05fc0
      Status: REGISTERED (valid)

  [3] Hash:   0x613700e7...f05fc0
      Chain:  Ethereum
      Type:   Authentication
      Issuer: did:key:z6MkDrJamesChen
      Parent: 0x5911f4d6...0e77e3
      Status: REGISTERED (valid)
```

```
  [4] Hash:   0x5911f4d6...0e77e3
      Chain:  IOTA
      Type:   Transfer
      Issuer: did:key:z6MkMeridianGallery
      Parent: 0xb48e3b1f...d02086
      Status: REGISTERED (valid)

  [5] Hash:   0xb48e3b1f...d02086
      Chain:  IOTA
      Type:   Origin
      Issuer: did:key:z6MkArtistElenaVasquez
      Parent: 0x00000000...000000
      Status: REGISTERED (valid)

======================================================================
  Result: VALID PROVENANCE CHAIN
  Links verified: 5
  Chains traversed: IOTA, Ethereum
======================================================================
```

B.2: Forgery Detection

```
======================================================================
  Forgery Detection Scenario
======================================================================

-> Registering forged transfer attestation on Ethereum...
   Type:    Transfer (sale)
   Issuer:  did:key:z6MkForger
   Parent:  0xbad0000000000000000000000000000000000000000000000000000000000bad
   Chain:   Ethereum

   Note: The contract accepts this registration because it does
   NOT validate parent hash existence (the parent could be on
   another chain). The forgery is detected during verification.

Registered. Gas used: 161552

-> Verifying forged attestation chain...

======================================================================
  Cross-Chain Forgery Detection
======================================================================

Walking chain from: 0x324a6abba21abd6816bd543cfe9b907fef226b43e231904bc62926ed3e2a4bbd

  [1] Hash:   0x324a6abb...2a4bbd
      Chain:  Ethereum
      Type:   Transfer
      Issuer: did:key:z6MkForger
      Parent: 0xbad00000...000bad
      Status: REGISTERED (valid)
```

```
  [!] Hash:   0xbad0000000000000000000000000000000000000000000000000000000000bad
      Ethereum: NOT FOUND
      IOTA:     NOT FOUND

========================================================================
  Result: BROKEN CHAIN
  Reason: Parent hash 0xbad00000...000bad not registered on
          any known chain.
========================================================================

========================================================================
  Summary
========================================================================

Painting scenario: VALID (5 attestations, 2 chains)
Forgery scenario:  DETECTED (broken parent link)

Attestations registered:
  IOTA:     3 (origin, gallery transfer, restoration)
  Ethereum: 3 (authentication, collector transfer, forged transfer)
```

This paper accompanies the Cross-Chain Asset Provenance Registry proof of concept, developed by Consensix Labs. The code is provided for research and educational purposes. It has not been audited for production use.