# VeriScript: A Protocol for Script Attestation on Cardano

VeriScript Project
Mark Petruska <mark.petruska@protonmail.com>, Adam Valach <avalach84@gmail.com>

December 23, 2025

**Abstract**

VeriScript is a protocol on the Cardano blockchain that enables users to verify both the source code and the on-chain binaries of scripts they interact with. It introduces a structured *attestation* mechanism based on native tokens, reference scripts, and script-addressed UTxOs to bind script hashes and addresses to human-readable metadata and signed claims from recognizable authors.

## Contents

## 1 Introduction

Smart-contract users on Cardano often interact with script addresses or policy IDs whose provenance and implementation details are opaque, making security reviews and trust assessments difficult.[1] VeriScript provides an on-chain attestation layer that lets authors and reviewers publicly link script binaries and identifiers to auditable source code and descriptive metadata.

The protocol defines:

- **Author tokens**, which identify attestation signers.
- **Attestation UTxOs**, script-addressed outputs that hold reference scripts, structured metadata, and non-reusable signature tokens.
- **Signature tokens**, which represent individual signers' endorsements of a particular attestation.

## 2 Design Goals

The main design goals of VeriScript are:

- **Verifiable provenance:** Bind a script hash/address/policy ID to a specific source-code revision (e.g. a GitHub commit) and minimal description.
- **Signer accountability:** Associate attestations with signers represented by author tokens controlled via minting policies.
- **Composability with Cardano primitives:** Use standard Cardano features such as native tokens, minting policies, script addresses, and reference scripts without protocol-level changes. [EUTxO][CIP-33][Minting]
- **Low overhead and reusability:** Allow multiple signers to attest to the same script via a shared attestation UTxO and non-reusable signature tokens, while respecting minimum ADA requirements. [Min-Ada]
- **Reclaimability:** Enable the original author to retire obsolete attestations and reclaim locked ADA by burning associated signature tokens.

## 3 System Model and Roles

### 3.1 Blockchain Environment

VeriScript is deployed on the Cardano blockchain, which uses an extended UTxO model.[2] Transactions consume existing UTxOs and create new ones, optionally involving Plutus scripts, minting policies, and reference scripts. [EUTxO][CIP-33]

The protocol assumes:

- Native tokens and minting policies as defined in the Cardano ledger. [Minting]

[1]Scripts on Cardano are associated with UTxOs at script addresses in the extended UTxO (EUTxO) model, and can be referenced via hashes or policy IDs. [EUTxO]

[2]Cardano UTxOs can be addressed by public keys or scripts, and may carry inline datums or reference scripts. [EUTxO][CIP-33]

- Reference scripts and reference inputs per CIP-31 and CIP-33 to store scripts on-chain in UTxOs. [CIP-31][CIP-33]
- Minimum ADA requirements based on UTxO size and asset content. [Min-Ada]

### 3.2 Actors

**Signer.** An entity capable of:

- Minting and controlling a unique *author token*.
- Building and submitting Cardano transactions that create attestation UTxOs or add their minted unique *signature tokens*.
- Verifying script source, compilation and derived identifiers off-chain.

**User (Verifier).** An entity that queries the VeriScript index/backend or directly the chain to find attestation UTxOs associated with specific scripts, then decides whether to trust them based on signers and statistics.

**Backend / Indexer.** Off-chain infrastructure that:

- Indexes attestation UTxOs and associated tokens.
- Exposes query interfaces based on script hash, address, policy IDs, or signer identity.
- Computes statistics (e.g. number of signers, signer reputation) to support trust decisions.

## 4 On-chain Objects

### 4.1 Author Token

An *author token* is a native asset minted under a dedicated minting policy that uniquely identifies a signer. The intended semantics are:

- One author token per signer (or signer identity), typically implemented as an NFT. [Minting]
- The minting policy enforces that a token can be minted only once.
- The token policy ID is used to verify ownership when minting signature tokens.
- The token implementation is similar to a CIP-68 token holding metadata that can potentially be used to link this digital identity to a real world one (if it is the choice of the signer). [CIP-68]

The author token serves as a capability: possession of it is required to mint signature tokens that represent attestations issued by the corresponding signer.

### 4.2 Signature Token

A *signature token* is a native token that represents one signer's endorsement of a specific attestation. It is minted under a *signature minting policy* which enforces:

- **Ownership check:** Minting a signature token requires demonstrating control of the associated author token. This can be enforced by requiring the author token to be moved in the same transaction, for example by sending it from the signer to themself.
- **Burnability:** Signature tokens can be burned under the same policy, allowing the original author of the UTxO or the original signer to burn the signature token.

### 4.3 Attestation Script and UTxO

VeriScript defines an *attestation script* whose address holds *attestation UTxOs.* Each attestation UTxO has the following structure:

- **Address:** The UTxO resides at the attestation script address.
- **Value:**
  - At least a protocol-compliant minimum amount of ADA, configured so that the UTxO can host multiple signature tokens without additional top-ups (e.g. $\geq$ 4 ADA, subject to actual min-ADA rules). [EUTxO][Min-Ada]
  - At least one signature tokens from various signers. The number of tokens a single UTxO can hold must be maximized to avoid the possibility of creating unspendable outputs.
- **Datum:** An *attestation datum* containing structured information that the signers attest to.
- **Reference script:** The actual script binary attached as a reference script per CIP-33. [CIP-33]

The attestation script enforces the allowed state transitions, including:

- Creation of new attestation UTxOs from arbitrary inputs.
- Addition of signature tokens to existing attestation UTxOs without changing the attestation datum.
- Spending and retirement of attestation UTxOs under controlled conditions (original author and signer only).

### 4.4 Attestation Datum

The *attestation datum* is an immutable record of what is being attested. It includes at least:

- A GitHub (or equivalent) reference including a repository locator and a specific commit hash, pointing to the source code.
- A minimal human-readable description of the script to help identification.
- The script hash of the compiled on-chain script.
- The script address, if applicable.
- The staking policy ID, if applicable.
- The minting policy ID, if applicable.
- The signer token name of the original author.

All signers that add signature tokens to the same attestation UTxO agree on the contents of this datum.

## 5 Protocol Workflows

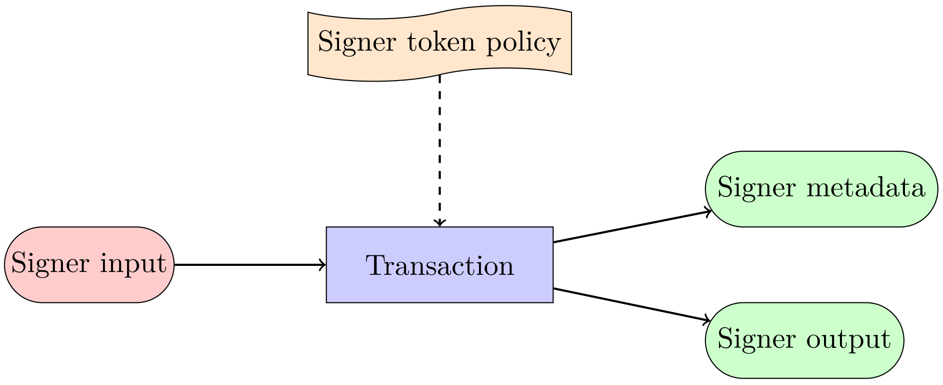
### 5.1 Signer Setup: Author Token Creation

This setup is performed once per signer identity:

1. The signer constructs a transaction that:
   - Mints exactly one author token under this policy.
   - Creates the corresponding "datum metadata" similarly to CIP-68.[CIP-68]
   - Sends the author token to an address controlled by the signer.
2. The transaction is submitted and included on-chain, finalizing the creation of the author token.

After this phase, the signer holds their author token and can participate in VeriScript attestations.

### 5.1.1 Transaction diagram



Where:

- **Signer input**: an input UTxO from the signer's wallet, covers the transaction fee and used in signer token name creation to ensure that the same name cannot be generated.
- **Signer token policy**: the minting policy that governs the minting and burning of the *signer token*s.
- **Signer metadata**: an UTxO stored at the address of the *signer metadata* script address, stores optional information about the signer.
- **Signer output**: an UTxO send back to the signer's wallet, contains the minted *signer token*.

## 5.2 Signer Attestation Creation (New Attestation)

When no suitable attestation UTxO exists for a script, the signer creates a new one:

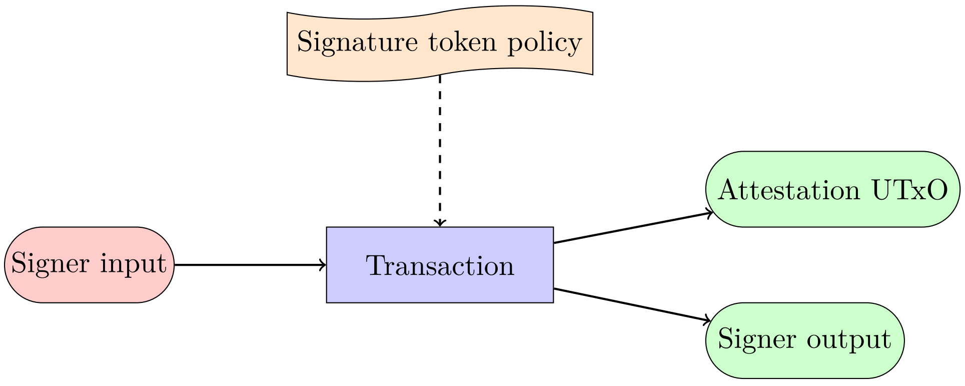
1. The signer obtains the script source code to be attested (e.g. by cloning a Git repository at a chosen commit).
2. The signer compiles the source code using the intended toolchain, deriving:
   - The script binary (e.g. flat or CBOR-encoded Plutus script).
   - The script hash.
   - The script address, if the script is used as a spending script.
   - The staking policy ID, if relevant.
   - The minting policy ID, if relevant.
3. The signer constructs an *attestation datum* containing:
   - The GitHub (or equivalent) repository and commit hash.
   - A minimal description of the script.
   - The script hash and any applicable script address or policy IDs.
   - Including his own author token's name
4. The signer builds a transaction that:
   - Creates a new UTxO at the attestation script address, with:
     - The attestation datum.
     - Sufficient ADA (e.g. 4 ADA, tuned to min-UTxO rules for a reference script plus future signature tokens). [EUTxO][Min-Ada]

    - One newly minted signature token under the signature minting policy.
  - Attaches the script binary as a reference script on this new UTxO. [EUTxO]
  - Demonstrates control of the author token, as required by the signature minting policy. This can be implemented by moving the author token in the transaction, for example sending it back to the signer's own address under specific constraints.

5. The transaction is submitted and finalized, creating a new attestation UTxO with the signer's initial signature token.

The resulting attestation UTxO serves as a canonical on-chain reference for that script, linking source, binary, and identifiers.

#### 5.2.1 Transaction diagram



Where:

- **Signer input**: an input UTxO from the signer's wallet that contains the *signer token.*
- **Signature token policy**: the minting policy that governs the minting and burning of the *signature tokens.*
- **Attestation UTxO**: an UTxO stored at the address of the *attestation holder* script address, stores the *attestation datum* described above. Contains the minted *signature token.*
- **Signer output**: an UTxO send back to the signer's wallet, contains the validated *signer token.*

### 5.3 Signer Attestation on Existing UTxO

If an attestation UTxO already exists for the target script, a signer can add their endorsement:

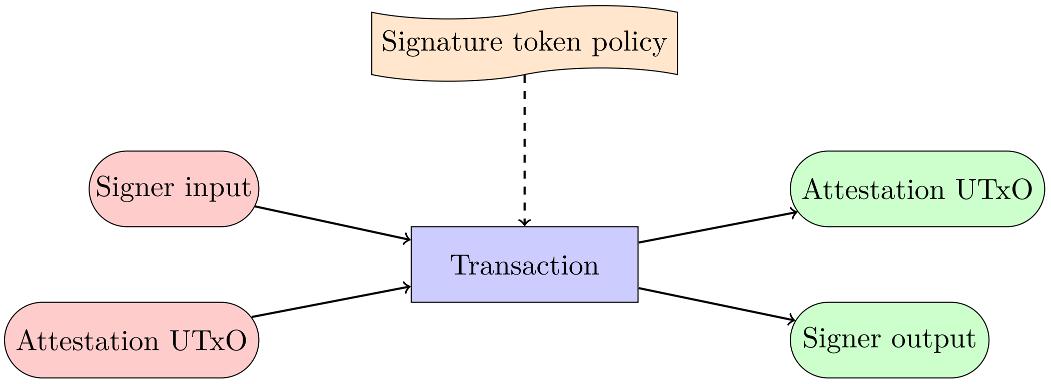
1. The signer locates the existing attestation UTxO via the VeriScript frontend/backend.
2. The signer independently:
   - Downloads the referenced source code (e.g. from the GitHub commit in the datum).
   - Recompiles and verifies that the resulting script hash, address, and policy IDs match the attestation datum.
3. If the attestation datum is correct, the signer constructs a transaction that:
   - Consumes the existing attestation UTxO.
   - Creates a new attestation UTxO with:
     - The *same* attestation datum.
     - The original ADA and signature tokens.

    - An additional signature token minted under the signature minting policy for this signer.
  - Proves control of the signer's author token as required by the signature minting policy.

4. The transaction is submitted and, upon inclusion, the attestation UTxO now reflects the extra signature token.

If the signer disagrees with the datum (e.g. the source does not reproduce the script hash), they instead create a new attestation UTxO as in the previous subsection, with corrected data.

### 5.3.1 Transaction diagram



Where:

- **Signer input**: an input UTxO from the signer's wallet that contains the *signer token.*
- **Signature token policy**: the minting policy that governs the minting and burning of the *signature tokens.*
- **Attestation UTxO** input: the UTxO that will be signed.
- **Attestation UTxO** output: the "new" *attestation UTxO* that has the same values as the original with the newly minted *signature token* attached.
- **Signer output**: an UTxO send back to the signer's wallet, contains the validated *signer token.*

## 5.4 User Verification Journey

A user who wants to verify a script performs:

1. The user queries the VeriScript backend (e.g. through the frontend application) or an indexer with a script hash, script address, staking policy ID, or minting policy ID to retrieve all relevant attestation UTxOs.
2. For each candidate attestation UTxO, the user can inspect:
   - The attestation datum (source reference, commit hash, description, script hash, addresses, and policies).
   - The set and count of signature tokens, each identifying a signer via its minting policy and token name.
   - Optional off-chain metadata or reputation scores about signers.
3. The user forms a trust decision based on:
   - The number and reputability of signers who have attested to this datum.
   - Any discrepancies across multiple attestation UTxOs referring to the same script identifiers.

- Other statistics provided by the VeriScript frontend (e.g. specific authors attestation numbers, counter-attested UTxOs, etc)

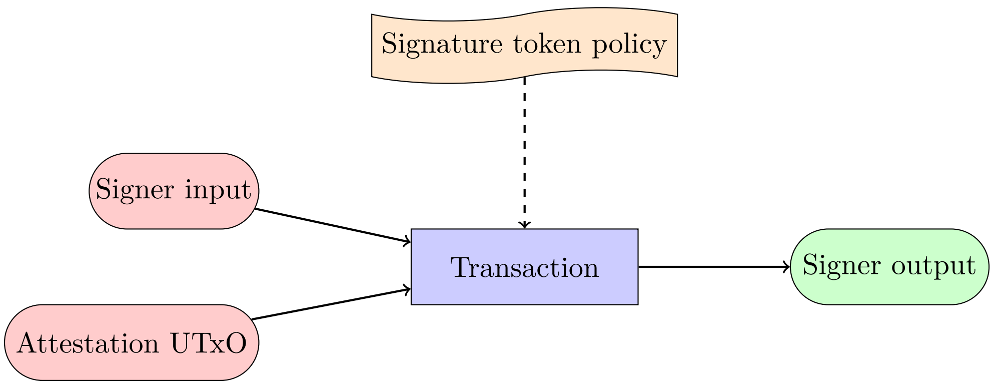
Users can, but are not required to, locally rebuild the script from the referenced source to further increase assurance.

## 5.5 Attestation Retirement

When an attestation UTxO is no longer needed (e.g. script deprecated or migrated), the original author may retire it:

1. The attestation script enforces a rule that allows the creator to spend the attestation UTxO.
2. The original author constructs a transaction that:
   - Consumes the attestation UTxO.
   - Burns all associated signature tokens under the signature minting policy.
   - Returns the locked ADA to an address of their choice.
3. Once the transaction is finalized, the attestation is effectively removed and the ADA is reclaimed.

### 5.5.1 Transaction diagram



Where:

- **Signer input**: an input UTxO from the signer's wallet that contains the *signer token*.
- **Signature token policy**: the minting policy that governs the minting and burning of the *signature tokens*.
- **Attestation UTxO**: the retiring *attestation UTxO*, all the signature tokens it contains must be burned.
- **Signer output**: an UTxO send back to the signer's wallet, contains the validated *signer token*.

# 6 Script and Policy Design

## 6.1 Attestation Script Logic

At a high level, the attestation validator should enforce:

- **Datum immutability for additions:** When an attestation UTxO is spent and recreated to add signature tokens, the output datum must equal the input datum.

- **Controlled retirement:** Spending that changes or removes the datum (i.e. not a simple signature-token addition) must satisfy additional constraints (e.g. possession of an author-control token).
- **Conservation of ADA and tokens:** Apart from explicit burns under minting policies, the ADA and existing signature tokens should be preserved when adding new signatures.

This logic can be implemented in a Plutus script that inspects:

- The input and output UTxOs at the attestation address.
- The datums and token bundles of those UTxOs.
- Redeemer fields specifying the intended action (e.g. "AddSignature" vs. "Retire").

### 6.2 Author Token Minting Policy

The author token minting policy should:

- Enforce that a unique token is minted for each author.
- Require the creation of a metadata (similar to CIP-68) during the minting operation.

### 6.3 Signature Token Minting Policy

The signature minting policy should:

- Require that the transaction demonstrates ownership of the corresponding author token by requiring the author token to appear as an input.
- Permit burning of existing signature tokens, in case the signer reconsiders his stance on an attestation or if the original author retires the attestation UTxO.

## 7 Incentives and Security Considerations

### 7.1 Incentive Structure

The primary incentive for signers is community benefit and reputational gain:

- Security auditors and experts can publicly attest to scripts, building a verifiable record of their reviews.
- Script authors can publish attestations for their own contracts, improving discoverability and transparency.
- DApp users can rely on aggregated attestations and signer reputations to inform interaction decisions.

Signers pay transaction fees and lock some ADA in attestation UTxOs, but can share UTxOs and reclaim ADA upon retirement.

### 7.2 Threat Model and Mitigations

Potential threats include:

- **Malicious or incompetent signers:** A signer may attest to incorrect or malicious scripts. Mitigation: rely on multiple independent signers, public reputation, and optional local verification of source and binaries.
- **Data inconsistency:** Different attestations might claim conflicting information about a script. Mitigation: users and backends can compare datums and highlight inconsistencies; signers can create corrected attestations when needed.

- **UTxO bloat:** Large reference scripts and many attestations could increase UTxO set size. Mitigation: limit the number of singature tokens that can be put on the same UTxO.
- **Author token compromise:** If a signer's key or author token is compromised, an attacker can mint fraudulent signature tokens. Mitigation: public announcements and possibly specific warnings appearing on the VeriScript frontend application.

## 8 Implementation Considerations

### 8.1 Reference Scripts and Min-ADA

Attestation UTxOs store the script binary as a reference script, allowing subsequent transactions to use the script without re-attaching it in the witness set, improving throughput and reducing fees. [EUTxO][CIP-31][CIP-33] Because reference scripts increase UTxO size, the minimum ADA requirement grows with script and token-bundle size, so the protocol recommends allocating generous ADA (e.g. 4 ADA) to accommodate multiple signatures over time. [Min-Ada]

### 8.2 Off-chain Infrastructure

A typical deployment includes:

- A backend indexer tracking:
  - All UTxOs at the attestation script address.
  - Their datums, reference scripts, and native tokens.
- APIs for:
  - Querying by script hash, address, policy ID, or signer.
  - Computing statistics such as number of attestations and distinct signers.
- Optional integration with GitHub or other hosting platforms to validate repository URLs and commit hashes.

### 8.3 Extensibility

Future extensions may include:

- Versioned attestation schemas enabling richer metadata (e.g. build instructions, dependencies).
- Support for non-Git platforms or content-addressed storage (e.g. IPFS).
- Reputation systems or staking mechanisms for signers.
- Formal verification artifacts (e.g. references to proofs) attached in the datum or via additional reference UTxOs.

## 9 Conclusion

VeriScript defines an on-chain attestation protocol for Cardano that links script binaries and identifiers to reproducible source-code references, while recording endorsements from identifiable signers. By combining author tokens, signature tokens, attestation UTxOs, and reference scripts, it enables transparent and composable provenance tracking for smart contracts and policies, without requiring changes to the underlying ledger.

## References

[CIP-31] Cardano Improvement Proposal 31: Reference Inputs (CIP-31). `https://cips.cardano.org/cip/CIP-31`.

[CIP-33] Cardano Improvement Proposal 33: Reference Scripts (CIP-33). `https://cips.cardano.org/cip/CIP-33`.

[CIP-68] Cardano Improvement Proposal 33: Datum Metadata Standard (CIP-68). `https://cips.cardano.org/cip/CIP-68`.

[EUTxO] Scripts and the Extended UTXO Model. Cardano / Plutus documentation. `https://plutus.cardano.intersectmbo.org/docs/essential-concepts/eutxo`.

[Min-Ada] Minimum ADA Value Requirement. Cardano Ledger documentation. `https://cardano-ledger.readthedocs.io/en/latest/explanations/min-utxo-mary.html`.

[Minting] Minting NFTs on Cardano. Cardano Developer Portal. `https://developers.cardano.org/docs/native-tokens/minting-nfts/`.