Wallet-Native Authentication for the Cardano Ecosystem

A Structured-Payload Sign-In Protocol Built on CIP-8, CIP-30, and CIP-93

Oscar Najera

Milestone 1 — Technical Document
Project Catalyst: Crypto Wallets for Signup, Login, and 2FA
https://arsmagna.xyz/project/authenticator

Contents

1 Introduction

Every meaningful relationship a person establishes with an online service begins with the same question: who are you? For three decades the answer has been a username and a password — a shared secret typed into a form and stored, in some form, by whoever is on the other end of the connection. This arrangement has produced a predictable and recurring failure mode: credential databases get breached, passwords get reused across services, and users carry the burden to manage an ever-growing pile of secrets they cannot realistically remember or protect.

Cardano wallets already solve a harder version of this problem. A wallet protects a private key that, if compromised, can drain real funds — and it does this well enough that millions of users trust it daily. The cryptographic machinery that secures a transaction signature is exactly the machinery needed to prove "I am the controller of this address" to a website, with no separate password, no separate database, and no separate secret to leak.

This document arises from the Catalyst-funded project "Crypto wallets for signup, login, and 2FA." It specifies, end to end, a protocol for wallet-native authentication on Cardano: what gets signed, how a server issues and validates a challenge, what a wallet should show its user before signing, and how the resulting verified identity serves more use cases than login. It is written to be directly implementable by developers and directly portable into a formal Cardano Improvement Proposal.

Critically, this is not presented as a green-field invention. Cardano already has the foundational primitives: CIP-30 standardizes how a webpage talks to a wallet, and CIP-8 standardizes how a wallet signs an arbitrary message. Currently the official Cardano documentation proposes to authenticate users by simply having them sign a nonce. CIP-93, correctly identified that signing an opaque nonce is insufficient and proposed a structured, human-readable payload instead — but it remains in *Proposed* status, with incomplete wallet-side adoption and no finished developer documentation. This document's contribution is to complete that picture: tightening the structured-payload approach, closing specific phishing and replay gaps, documenting the off-chain authorization advantages unique to Cardano's native-token model, and packaging all of it so that adoption is a matter of reading one document, not assembling fragments from five sources.

1.1 Document Scope

This document covers four layers, each building on the last:

1. The wallet interaction layer — how a dApp and wallet communicate, building on CIP-30.
2. The message layer — the exact structure of the signed payload, extending CIP-93.
3. The verification and session layer — server-side validation, nonce lifecycle, and audit logging.
4. The authorization layer — how a verified address becomes an off-chain entitlement check against on-chain token holdings.

Implementation libraries, language-specific code, and the education/advertising materials are covered by later milestones and are out of scope here; this document defines the protocol they will implement.

2 Justification

Three independent lines of reasoning justify this work: a security argument, an adoption-gap argument, and a differentiation argument. Each is addressed in turn.

2.1 The Security Argument

Password authentication fails in a specific, well-understood way: the server must hold a secret (a password, or a salted hash of one) that, by itself, is enough to impersonate the user. Any breach of that store compromises every user in it, retroactively and permanently, since passwords are commonly reused across services.

Wallet-based signing removes the shared secret entirely. The server stores only a public verification key and an address — information that is, by design, safe to leak, because it confers no ability to forge a new valid signature. Authentication becomes proof of possession of a private key, demonstrated freshly on every login via a signature that is unique to that login attempt and cannot be replayed.

However, a security argument is only as strong as its weakest implementation detail, and the weakest detail in most current Cardano implementations is the content of the signed message itself.

2.2 The Implementation Gap: Why "Sign a Nonce" Is Not Enough

The Cardano Developer Portal's current reference pattern instructs a server to generate a random nonce and have the wallet sign it. This is a reasonable minimum, but it leaves three gaps open:

- **No domain binding.** A signed opaque string carries no information about which website requested it. If a phishing site can obtain a valid nonce signature — for instance by relaying a real login request from a different, legitimate site — nothing in the signed payload itself reveals the mismatch.
- **No user-readable intent.** A wallet asked to sign a random string presents the user with exactly that: a meaningless string of characters. There is no way for the user to distinguish a benign login request from a malicious payload, because both look identical: unreadable. This is the precise fear that drives user hesitation — "what if I am tricked into signing something that empties my wallet?" CIP-8 signing is not a transaction and cannot move funds, but the user has no way to verify that for themselves from the wallet prompt alone.
- **Thin audit trail.** A bare nonce, once verified, is consumed and discarded. The server's authentication log records only that some signature, valid at some time, was checked. It cannot reconstruct what the user was shown, what action they consented to, or which endpoint the credential was scoped to.

CIP-93 ("Authenticated Web3 HTTP Requests"), proposed in December 2022, already identifies this exact problem: a static, opaque signing target is dangerous because an intercepted signature could be misused, and the fix is a dynamic, structured, human-readable payload rather than a bare string. It is the right design. It has not, however, reached the level of wallet support or developer documentation needed for ecosystem-wide adoption — fewer than 80% of wallets currently implement its recommended parsing and domain-checking behavior. This document extends CIP-93's structured-payload approach as its message layer, closes several specific gaps in it (Section 4), and treats promoting its adoption path as a responsibility of this project.

2.3 Differentiation: Why Build This Rather Than Use Passkeys

WebAuthn and platform passkeys deserve direct acknowledgment because their standardization which makes them the strongest competing approach, and being silent about that would weaken this proposal's credibility. Passkeys achieve a security property this protocol cannot fully match by construction: origin binding enforced by the browser and operating system, not by convention. A passkey credential is cryptographically bound to a relying party (RP) ID. The browser ensures that only the current site's RP ID can be requested, and the authenticator will only use a credential whose stored RP ID matches that request. As a result, a phishing site cannot obtain a valid signature using another site's passkey, regardless of whether the user is fooled by the page's appearance.

This protocol cannot claim that property, and should not claim it. CIP-30 wallets are not required to enforce origin binding; at best, a wallet can choose to check it and warn the user,

voluntarily. The honest case for this protocol is not that it beats passkeys on phishing resistance. It is two other things:

- **Asset-backed authorization, not just authentication.** A passkey proves who you are and nothing more. A verified Cardano address is simultaneously proof of identity and a live query key into the holder's on-chain native tokens — enabling token-gated access, membership tiers, and reward eligibility checks with no smart contract and no separate identity system. Section 5 develops this in detail.
- **Reuse of infrastructure people already trust with real value.** Cardano users already protect a wallet that secures actual funds, with backup and recovery practices shaped by that fact. This protocol asks them to reuse that existing discipline for authentication, rather than create and manage a new, platform-locked passkey credential per device.

This protocol and passkeys are not mutually exclusive; a service can reasonably offer both. The case for this project is that the Cardano ecosystem currently has no well-specified, ecosystem-endorsed version of the wallet-signing option, and building one well captures advantages passkeys structurally cannot offer.

3 Protocol Overview

At a high level, authentication proceeds through four steps, summarized below and detailed in the sections that follow.

Protocol Flow at a Glance

1. **CONNECT** — The dApp requests access via CIP-30's `enable()`. The wallet records the requesting origin.
2. **CHALLENGE** — The dApp asks the server for a nonce, scoped to an address and an action. The server returns a structured challenge.
3. **SIGN** — The wallet assembles the structured CIP-93-style payload, displays it in human-readable form, and signs it via CIP-8 / CIP-30's `signData()`.
4. **VERIFY** — The server checks the COSE signature, the address binding, the domain, the nonce, and the timestamp window, then issues a session and writes a full audit record.

3.1 Layered Design

Table 1: Standards relied upon at each layer, and this document's contribution

Layer	Standard Relied On	What This Document Adds
Wallet ↔ dApp transport	CIP-30	Specifies stake-address use and reuses connection-time origin for binding (§4.2)
Message signing primitive	CIP-8 (COSE_Sign1)	No changes; used as-is
Message structure	CIP-93 (Proposed)	Tightens domain enforcement, adds server-committed action, clarifies address/key selection (§4)
Login semantics and session	None existing — original to this project	Nonce lifecycle, verification steps, audit logging (§4.4–4.6)
Authorization beyond login	Cardano native tokens (no smart contract required)	Off-chain entitlement model using the verified address (§5)

4 Message Format and Verification

4.1 Signed Payload Structure

Building directly on CIP-93's JSON payload shape, the signed message is a structured, human-readable object — never a bare random string. The wallet is expected to render this payload's contents to the user before requesting their approval, not present it as an opaque hex blob.

Table 2: Fields of the signed authentication payload

Field	Required	Purpose
`uri`	Yes	Full endpoint path the credential is destined for, including hostname. Used for domain binding (§4.2).
`action`	Yes	Human-readable purpose, drawn from a small fixed vocabulary where possible (§4.3).
`actionText`	No	Localized / extended description of the action, shown verbatim to the user.
`address`	Optional	Echoes which address/key the wallet is signing with, for display in the confirmation prompt; the signature is already cryptographically bound to an address by CIP-8/CIP-30 regardless (§4.5).
`nonce`	Yes	Server-issued, single-use random value (§4.4).
`timestamp`	Yes	Time the wallet is signing, refreshed by the wallet immediately before signing (§4.4).

4.2 Domain Binding: Reusing the CIP-30 Connection Origin

CIP-93 asks wallets to check the payload's `uri` field against an allow-list, but leaves open where that allow-list comes from. This document specifies a concrete source: the same origin the wallet already captured when the dApp called the CIP-30 `enable()` entrypoint to establish the connection in the first place.

Concretely: if a signing request's `uri` does not match the origin that the connected session was established under, the wallet should refuse the signing request outright, not merely display a warning. This converts an easily-skipped warning into a hard constraint, using information the wallet already possesses from the existing CIP-30 handshake — no new wallet infrastructure is required, only a stricter check on data already collected.

4.3 Human-Readable Intent and Visual Differentiation

To directly address the user fear that a malicious payload could be disguised as an innocuous login and used to deplete a wallet, this document specifies a two-tier presentation rule for wallets:

- **Recognized actions.** If the `action` field matches a small, fixed vocabulary the wallet recognizes ("Sign in", "Sign up", "Reauthenticate"), the wallet renders a standard, calm login prompt.
- **Unrecognized actions.** Any payload whose `action` falls outside that vocabulary is rendered in a visually distinct, higher-friction warning state — different color treatment and an explicit "this is not a standard sign-in request" banner — rather than being displayed identically to an everyday login.

This does not, and cannot, prevent a user from approving something harmful if they choose to ignore the warning. What it removes is the specific failure mode of an unusual request being visually indistinguishable from a routine one.

4.4 Nonce Lifecycle and Replay Protection

1. Server generates a cryptographically random nonce on challenge request, bound to the requested address and a server-chosen maximum freshness window (recommended default: 5 minutes, adjustable per service).
2. Wallet refreshes the `timestamp` field immediately prior to signing, so the signed time reflects actual signing time, not challenge-issuance time.
3. Server marks the nonce consumed immediately upon first successful verification. A second presentation of the same nonce is rejected unconditionally, regardless of signature validity.
4. Server rejects any payload whose `timestamp` falls outside the freshness window, independent of the nonce check.

4.5 Address Selection: Stake Key as Default, With Room to Deviate

CIP-8 and CIP-30 already bind a signature to an address cryptographically, via the COSE_Sign1 protected headers and accompanying COSE_Key — this binding does not depend on the JSON payload. The payload's `address` field instead serves the wallet's confirmation prompt: showing the user, alongside `action` (Section 4.3), which account is about to sign.

Stake address is the recommended default for plain sign-in, since it stays stable across payment-address rotation. It is not mandated: a service may need a specific payment key instead — for example, proving control of a particular UTXO-holding address, authenticating as one sub-account in a multi-account wallet, or DRep/governance-key signing under CIP-95. Whichever address type an endpoint expects, the server should be explicit and consistent about it.

4.6 Server-Committed Action

The `action` field is not just wallet-display vocabulary — it is also what the server checks the signature against. The nonce-issuance response commits to a specific `action` value; the wallet

signs that same value; the server verifies the signed `action` matches what it issued. This closes a real gap: without it, nothing stops a server from accepting a signature labeled "Sign in" as proof of a different, more sensitive action than the user understood they were authorizing. With it, the signature is cryptographically tied to the specific purpose the server declared at issuance, not to whatever the server later chooses to use it for.

4.7 Server-Side Verification Checklist

On receiving a signed payload, the server performs the following checks, in order, rejecting on the first failure:

1. Parse the COSE_Sign1 structure (CIP-8) and recover the protected headers and signed payload.
2. Recompute the address from the recovered public key and confirm it matches the address claimed for this request (the specific address/key type expected by the endpoint, per Section 4.5).
3. Confirm the nonce is known, unconsumed, and was issued for this address.
4. Confirm the timestamp falls within the freshness window.
5. Confirm the `uri` matches the expected endpoint and domain.
6. Confirm the `action` matches what was committed at nonce-issuance time.
7. Verify the COSE signature itself against the recovered public key.
8. On success: mark the nonce consumed, issue a session, and write a full audit record (§4.8).

4.8 Audit Logging

Because the full structured payload — not a bare nonce — is what gets signed, the server can retain the entire verified payload as its audit record: which address, which server-committed action, which endpoint, and the time the wallet itself attested to signing. This is a strictly stronger record than a password-based login log, which can only show that some secret string matched at some time. A leaked copy of this audit log confers no ability to forge a future login, since reproducing a past signature does not allow constructing a new valid one.

5 Beyond Login: The Authorization Layer

A verified address is more than proof of identity. Because Cardano native tokens are first-class ledger objects — unlike Ethereum's ERC-20 tokens, they require no smart contract to exist or be queried — a server holding a cryptographically verified address can cheaply check what that address actually holds, and use that as an additional, live authorization layer. This is the central advantage this protocol can offer that platform passkeys structurally cannot, since a passkey is not attached to any asset. For this authorization use case specifically, the stake address is usually the most natural choice (Section 4.5), since token holdings and governance participation are typically tracked at the account/stake level rather than per payment address.

5.1 Use-Case Catalogue

Table 3: Off-chain authorization use cases enabled by a verified address

Use Case	How It Works	Why It Needs No Smart Contract
Passwordless login	Wallet ownership itself is the credential; no separate password exists to store or leak.	N/A — base case
Token-gated content	Server checks the verified address's holdings for a specific native token or NFT before granting access.	Native tokens are queryable ledger state directly
Membership tiers / perks	Different token holdings map to different perk tiers (discounts, early access, vouchers).	Tier logic lives entirely off-chain, in the server's own business rules
Reward / airdrop eligibility	Server confirms the authenticated address held a qualifying asset at a relevant time before releasing a reward.	No on-chain claim contract required for eligibility checks
Whitelist verification	Server matches the verified address against a pre-registered allow-list (e.g. for a minting window).	Allow-list is a simple off-chain set membership check
Approving off-chain actions	A signed, scoped payload authorizes a specific off-chain action (e.g. confirming an in-game trade) without an on-chain transaction.	Action authorization is separated from settlement

5.2 Why This Is a Genuine Cardano-Ecosystem Advantage

This is not a generic blockchain talking point; it follows specifically from how Cardano's native multi-asset ledger is built. A backend integrating this protocol can perform an entitlement check as a simple, cheap chain query against the verified address — no custom contract deployment, no gas-metered call, no separate identity bridge. The signature that authenticates the user and the token balance that authorizes their access both rest on the same address, queried the same way.

There is also a composability advantage worth advertising on its own: because the stake address is the same identity surface used for staking delegation and on-chain governance participation, a service can — if it chooses — connect a user's authenticated session to other on-chain reputation signals (delegation history, governance voting record) that a freshly minted passkey credential, by design, has no relationship to whatsoever.

6 Comparison with Alternative Approaches

The table below situates this protocol honestly against its closest alternatives. Stronger does not mean strictly superior in all respects — each approach makes different trade-offs, and the right comparison is what each protocol enforces structurally versus what it merely recommends.

Table 4: Protocol comparison across domain binding, intent legibility, replay protection, and asset awareness

	Raw Nonce (current Cardano default)	This Protocol (CIP-93-based)	SIWE (Ethereum)	WebAuthn / Passkeys
Domain binding	None	Voluntary, wallet-checked	Voluntary, wallet-checked	Enforced by OS/browser
User-readable intent	None	Yes, structured + tiered warnings	Partial (statement field)	Not applicable — no message
Replay protection	Nonce only	Nonce + timestamp + committed action	Nonce + timestamp	Single-use challenge
Asset-aware authorization	No	Yes — native, no smart contract	Requires separate ERC-20/721 calls	No — not asset-linked
Adoption status (Cardano)	De facto default today	Specified, needs completion	N/A (Ethereum-native)	Growing, enterprise-driven