A valid proof cannot speak about what was never submitted.

Utuh is a completeness layer for the Attestcoin Protocol, and the undercollateralized credit line it exists to serve.

THE GAP

Attestcoin proves a transaction happened. It cannot prove a set of them is complete.

Whoever submits the proofs chooses which proofs to submit — and every one of them verifies. A registry built on inclusion proofs can tell you what it holds. It cannot tell you that what it holds is *all of it*.

THE SENTENCE CREDIT NEEDS

“This borrower has **never** been liquidated.”

A statement about events that do not exist. No number of inclusion proofs reaches it, and the borrower is not going to submit the one that would.

NOT HYPOTHETICAL

Creditcoin's own reference loan flow settles its one adverse outcome with a key.

```
// gluwa/attestcoin-protocol-examples @ 6668487ad07f
// loan/contracts/sol/ASCLoanManager.sol

function _noteLoanRepayment(bytes memory tx_) internal
    // ← proven through 0x0FD2

function markLoanAsExpired(uint256 loanId)
    external onlyOwner
    // ← an administrator decides
```

Every *positive* fact is cryptographic. The one negative fact — a borrower who did not repay — is an owner call, because default is the absence of a transaction and there was nothing else to settle it with.

WHY THIS IS THE RIGHT EXAMPLE

It is not a flaw in a tutorial. It is the protocol's shape showing through, in the most neutral code available — the reference every builder in this ecosystem starts from.

`docs/COMPLETENESS.md` is the same reading as three questions anyone can run against their own contract.

THE MECHANISM

Verified on the way in. Bonded on the way out. Broken by one proof.

01 Every member of a claim is verified by the Block Prover before it is stored, so a claim **can never be padded**.

02 What it can be is **short**. So the claimant posts a bond against the assertion that nothing was left out.

03 Anyone who proves **one in-scope event the claim omits** voids it and takes half the bond. The registry never verifies a whole set — one proof settles a claim, or none does — so a claim spanning thousands of events stays cheap.

> Presence stays cryptographic. Absence becomes economic.

LIVE, NOT A MOCKUP

A claim sealed one event short, broken from a browser with no backend.

The console at /app/ sweeps Ethereum across independent endpoints, takes the union rather than a vote, checks every event against the claim on-chain, and sends the refutation itself. No server, no key to look, no indexer. Four files.

short claim	utuh.vercel.app/?claim=5
false clean	?deployment=mainnet&claim=20

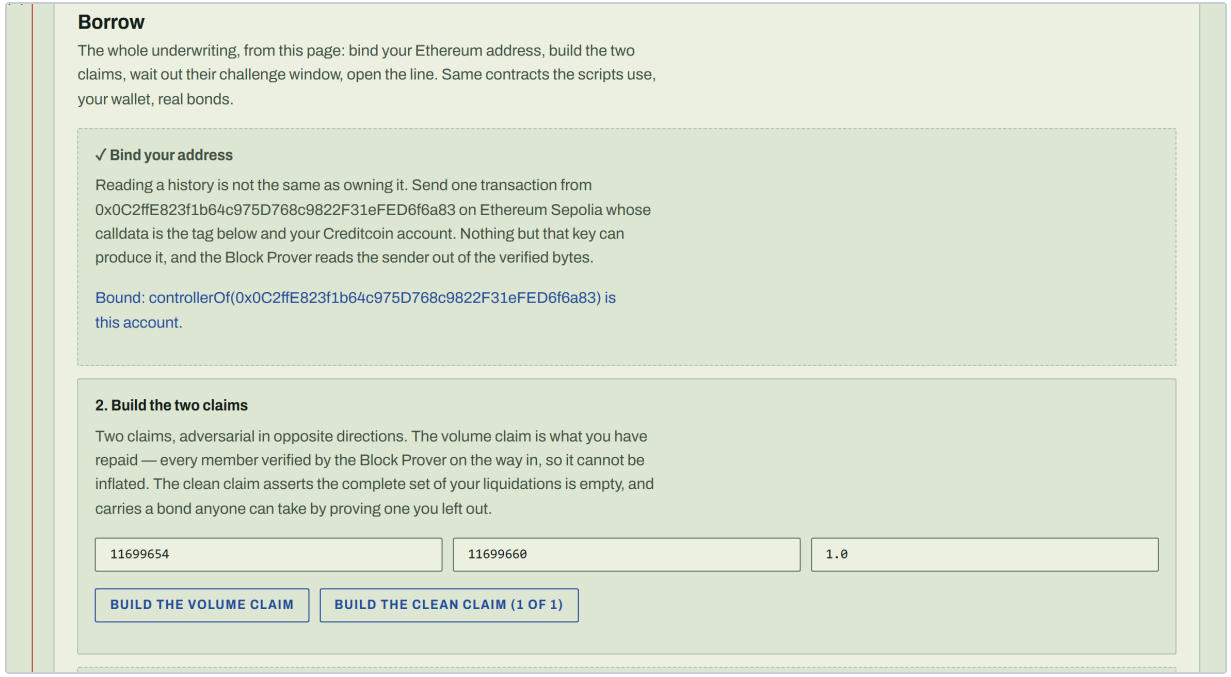
Both open the console on the claim, verdict first. Asserted hourly against the published build, so a dead link is red rather than quiet.

Undercollateralized CTC credit, underwritten on real Ethereum history.

Proven repayment volume, plus a bonded claim that the borrower was never liquidated. Nothing bridges: the history stays on Ethereum, the credit is issued on Creditcoin, repayment happens back on Ethereum, and only proof crosses.

Reading a history is not owning it — the borrower binds their address with one transaction whose calldata only their key could produce, and the Block Prover reads the sender out of verified bytes.

`Ethereum mainnet, chainKey 3` `not Sepolia`



THE CEILING

What lends is the guarantee, not the volume.

849 CTC	10 CTC
what proven volume alone would justify	what the line actually opens at

The limit is `enforceableLoss` × the bond multiple, not the history. A claim is only worth what someone could have taken by breaking it — so a large history behind a small bond underwrites a small line, and says so.

AND THEN IT REFUSES

Pointed at a stranger's history, `npm run credit` stops at `SubjectNotControlled` and opens nothing. The refusal is the demonstration: everything up to it was read off a public chain by somebody who does not own that address.

All sixteen entry points — and what each of them is for.

BLOCK PROVER · 0X0FD2 · FIVE

Both `verifyAndEmit` overloads — the batch form for appending, the single form for a refutation, which only ever needs one. `calculateTxIndex` for the ordering key, taken from the proof rather than from the caller. The decoder for scope matching against verified bytes, and the `view` twin over `eth_call`, which is how the whole path was mapped with an empty wallet.

CHAININFO · 0X0FD3 · ELEVEN

The attestation gate that makes challenge windows sound, the frontier that bounds staleness, the genesis that bounds a range, the bounds that say whether a height is provable *yet*, and the digest inversion that lets a browser check the indexer against the chain itself — a third leg on the oracle audit that nothing else here could have supplied.

A count is the weak half of the argument. Seven of the eleven were added when a stocktake found them unused, and each took a real job — none of it is a call made to be counted.

THE FAIR OBJECTION

A bond only deters if somebody looks. One participant does not need paying to.

```
UtuhCredit.openLine
  → UtuhRegistry.isUsable(claimId, exposure)
      → status == Finalized
          // the challenge window has closed
```

So that window *is* the lender's diligence window, and the money a false clean claim takes is the lender's own. Half the bond is a rebate on work they had to do — not a wage that has to clear.

AND THE HONEST BOUNDARY

This makes watching rational for lenders. It does not make it funded as a public good: a claim nobody intends to lend against still finalizes with nobody looking. For credit that is the right shape. For a general-purpose fact registry it would not be.

Which is why the console needs no backend, and why `npx utuh-mcp` puts the same role behind an agent. Watching should cost a tab, not a team.

WHERE THE MONEY IS

One choke point, and nothing charged yet.

`openLine` is where a lender turns two bonded claims into a limit — the only place a number exists that could not have existed without this layer.

That is where a production lender pays, in basis points on the limit opened. Watchers are paid by the mechanism rather than out of revenue, from bonds that lying claimants posted. Borrowers pay nothing to be measured: the history is already theirs and already public.

WHAT IT DOES NOT NEED

A token. Any bounty large enough to matter is recoverable by a claimant refuting their own claim from a second address — the same front-running that made `enforceableLoss` necessary. A token layered on that would be worse than the honest gap.

A fee on a testnet demo is theatre, so there isn't one. The seam is named instead.

WHAT IS REAL, AND WHAT IS NOT

The limits, before anyone has to find them.

ECONOMIC, NOT CRYPTOGRAPHIC

A bond makes lying expensive, not impossible. And one finalized claim can back a line at every lender at once.

READ-SIDE ONLY, FOR NOW

Attestcoin writability is in third-party audit and not on testnet. A default is recorded on Creditcoin; enforcing it back on Ethereum waits. The seam is reserved in the design.

CLAIM SIZE HAS A CEILING

Members are a storage array, so refutation is a binary search the chain runs itself with no witness a claimant can withhold. Past ~10,000 events that stops being affordable; the replacement — an incremental Merkle root with an adjacency proof — is built on a branch, 165 tests in CI, not deployed.

THERE IS ONE OF US

The bus factor is one. What limits it is that every claim here is checkable without asking anyone.

MEASURED, NOT NARRATED

All of it is on a public chain, and none of it was written by us.

212	84	34	34 CTC
events proven into claims, through the Block Prover	claims sealed on the two published registries	broken by a proof of an omitted event — all from four keys this project holds; no outside user yet	bond actually slashed, not modelled

211 Foundry tests, 100% branch coverage, 99.1% of mutants killed, 121 guards asserted live, Slither at zero findings across 97 detectors. 376 transactions and 156.6M gas into deployed contracts, every one verified on Blockscout and matched on Sourcify. CI re-proves the two proof builders agree byte-for-byte daily and smokes the published console hourly.

site	utuh.vercel.app · console /app/
code	github.com/PugarHuda/utuh
watcher	npx utuh-mcp · 0.4.0
oracle	dashboard.cc3-testnet.creditcoin.network