Arrears

A failed Ethereum transaction, admitted as evidence against a bonded operator.

A real failure. A bonded operator. Turned away.

It failed inside the covered window, on a covered contract. Arrears refused to touch the bond, because the operator never promised to answer for `withdraw()`.

THE CLAIM, CHECKED AGAINST THE PROMISE

OPERATOR	`0x9733EcE9…6B178E` bonded on Creditcoin	✓
CHAIN	Ethereum Sepolia Attestcoin chain key 1	✓
WINDOW	11,646,000 – 11,700,000 block 11,646,965 is inside	✓
CONTRACT	`0xfFf99767…4d6B14` WETH9 — covered	✓
SELECTOR	`0x2e1a7d4d` withdraw(uint256) — in no coverage	✗

RULING · TURNED AWAY

```
OutOfScope(Selector, 0xfFf99767…4d6B14,
  0x2e1a7d4d, 11646965)
```

mined on CC3 `0xd9f96284…a95685`
the failure `0x5c02af74…c3f251`

A rule that only ever says yes is not a rule.

Any contract can move money when it is handed a proof. What makes Arrears a court rather than a payout button is what it declines — on chain, with a named error, in an explorer nobody here controls.

89.7%

of reverted mainnet transactions are explicit reverts — the contract said no on purpose. Arrears must record every one and slash none of them.

`647 of 721 · measured, 200 attested blocks`

1 · 1 · 2

Four mined rulings. One moved a bond, one kept a record, two turned the claim away. None of them is an error.

`bond moved · record kept · turned away`

0

claims written by the strict path when it refused. It declined and left no trace — demonstrated on chain, not described in a spec.

`claim.ruledAt == 0 after refusal`

Can a *reverted* Ethereum transaction be proven on Creditcoin?

ETHEREUM MAINNET · GROUND TRUTH

TX	0x06ba12d8eaf7527634e9739dc42b778cd2b60d9976901930233f6401e9042dd7
BLOCK	25,916,354 · index 85
STATUS	0x0 — reverted
GAS USED	386,677 of 598,875
LOGS	0

ON CREDITCOIN CC3

PRECOMPILE	0x…0FD2 verifySingle → true
DECODER	decodeReceiptFields → receiptStatus 0

Yes.

The proof verifies, and the failure is readable on chain. Nobody had done it; it is the entire premise.

AND IT IS ALREADY EXPLOITABLE

A naive contract accepted this exact failure as a completed settlement:

`0xbd4eedc2…393c52`

Its strict twin refuses the same proof with `SourceTransactionReverted(0, 386677, 598875)` — every value decoded from the proven bytes.

A reverted transaction has no logs. Ever.

813 / 813

reverted mainnet transactions with zero logs and an all-zero bloom, across two independent scans. That is EVM semantics, not sampling: a top-level revert rolls back the journal, logs included.

The documented Attestcoin pattern is event-driven — prove a transaction, read its logs. That pattern doesn't reach failures, because a failure has none.

It isn't a limit of the protocol. The decoder reads `from`, `to`, calldata and `status` on any proven transaction, for anyone who thinks to ask. **We were first, not alone.**

PROVEN, FROM THE BYTES THE PRECOMPILE VERIFIED

FIELD	WHAT IT TELLS ARREARS
`receiptStatus`	that it failed — `0`
`gasUsed · gasLimit`	whose fault it was
`to`	which contract
`selector`	which promise
`from`	whose bond
revert reason	not committed — never recoverable

Revert return data is not part of what Attestcoin commits to. Arrears is built entirely on fields that are.

Reverting is not failing a duty. Running out of gas is.

ETHEREUM MAINNET · 200 ATTESTED BLOCKS

transactions	49,140	measured
reverted	721 · 1.47%	measured
per day	~26,000	extrapolated
explicit revert	647 · 89.7%	record, never slash
out of gas	74 · 10.3%	slashable

Reverting is normal: a swap that moved, a check that failed. Punishing it would punish participation.

ARREARSVERDICT.SOL — THE WHOLE RULE

```
function classify(uint8 receiptStatus,
                  uint64 gasUsed, uint64 gasLimit)
    internal pure returns (Verdict)
{
    if (receiptStatus == 1) return Verdict.Succeeded;
    return gasUsed >= gasLimit
        ? Verdict.OutOfGas
        : Verdict.ExplicitRevert;
}
```

The sender chose the limit. Nobody can race them into choosing it badly, so it is the one failure that is unambiguously theirs.

`internal`, so the compiler inlines it into both the court and the probe: the gallery and the slash run the same bytes.

A refusal is the system working.

OUTCOME 1 OF 3

Bond moved

out of gas — the sender chose the limit

as at CC3 block 5,440,296

bond	-2.0 tCTC
treasury	+2.0 tCTC
credit limit	1,000 → 750
premium	500 → 650 bps
strikes	0 → 1
claims on record	+1

0xa7b1e50e…976be2

WETH9 deposit() · 30,000 of 30,000 gas

OUTCOME 2 OF 3

Record kept

explicit revert — on the record, never slashable

as at CC3 block 5,440,310

bond	unchanged
treasury	unchanged
credit limit	unchanged
premium	unchanged
strikes	unchanged
claims on record	+1

0xe585da11…cda6c1

NotSlashableExplicitRevert · 0x6bf93631

OUTCOME 3 OF 3

Turned away

outside the promise — nothing happened at all

as at CC3 blocks 5,440,311 · 5,440,353

bond	unchanged
treasury	unchanged
credit limit	unchanged
premium	unchanged
strikes	unchanged
claims on record	0

0xd9f96284…a95685 out of scope

0xc0bf98c0…4463c7 strict path

DEMONSTRATED, NOT DESIGNED

claim.ruledAt == 0

The strict path refused an explicit revert and wrote nothing. After the refusal, the claim that would have existed still reads as never ruled.

submitClaim

Records every proven failure. Slashes only if it is slashable. An explicit revert lands as Record kept.

submitSlashingClaim

Slash or nothing. An explicit revert reverts the whole call — no record, no strike, no state.

THE STRICT PATH, ON CHAIN

THE FAILURE	0xdc8730cf…c1b933
GAS USED	24,187 of 100,000
SUBMITTED	0xc0bf98c0…4463c7
ERROR	NotSlashableExplicitRevert(24187, 100000)
RECORDED	nothing — ruledAt 0

The reason is read from `previewClaim` over `eth_call`. On any EVM chain, a mined revert never hands its reason back through the node — so the preview is the one moment the reason is in hand, and it is read there.

CLASSIFICATION AT REAL SCALE

Seven mainnet failures, three years.

STRESS WINDOW	BLOCK	GAS USED / LIMIT	VERDICT	TRANSACTION
USDC depeg (SVB) 2023-03-11	16,806,527	324,239 / 324,239	OutOfGas	0xc22eb305…2a8096
USDC depeg (SVB) 2023-03-11	16,806,523	77,600 / 77,600	OutOfGas	0x1dd76820…e71c22
Yen carry unwind 2024-08-05	20,462,242	134,138 / 134,138	OutOfGas	0x252a53c5…6625e4
Yen carry unwind 2024-08-05	20,462,236	76,808 / 76,808	OutOfGas	0xbf4a6412…a206d0
Feb 2025 selloff 2025-02-03	21,769,440	134,482 / 134,482	OutOfGas	0x3198a097…224fcc
Oct 2025 cascade 2025-10-10	23,549,876	120,000 / 120,000	OutOfGas	0x27cb5855…4a967c
Oct 2025 cascade 2025-10-10	23,549,876	80,000 / 80,000	OutOfGas	0xee76fbbb…f3c756

Each proven against the live precompile and classified by the same inlined `ArrearsVerdict` that slashes on Sepolia. **No bond attaches** — nobody here holds those keys, so they stand as a record of what the protocol would have caught.

```
$ cd demo && npm run gallery
re-proves all seven live · no key, no funds
```

SCOPE IS THE PRODUCT

A bond answers for a promise, not for everything.

THE DEMO OPERATOR'S DECLARED COVERAGE · SEPOLIA

CONTRACT	SELECTOR	BLOCKS
WETH9	deposit()	11,646,000 – 11,700,000
WETH9	transfer(address,uint256)	11,646,000 – 11,700,000
WETH9	approve(address,uint256)	11,646,000 – 12,000,000
WETH9	withdraw(uint256)	in no coverage

Coverage is a (contract, selector) set plus a window. Where several cover a claim, the court takes the widest payable, ties to the earliest declared.

OUTOFSCOPE NAMES THE AXIS THAT MISSED

Operator · Chain · Window

Target · Selector

Revoked · Expired · Exhausted

Revocation stops future cover. It never removes past liability. A failure at or below the revocation height stays claimable until the deadline passes — tested on both sides of the boundary.

UNDOCUMENTED

The batch cap is ten.

Both `verify` and `verifyAndEmit` take up to 10 proofs. The 11th reverts with `heights: Value is too large for length`. A protocol bound, not a gas one — ten used 0.515% of the block cap.

A FOOTGUN IN THE DOCUMENTED PATTERN

Nobody checks the emitter.

The precompile proves inclusion and makes no claim about who emitted a log — correct behaviour. But matching an event by signature invites a check nobody performs. A 279-byte contract on Sepolia emitted a forged 1,000,000 USDC `Transfer`; a naive contract credited it, with `emitterWasExpected = false` in the same event.

accepted	`0x7d81c702…c56947`
impostor	`0xfC7eAbb2…7CAcB8`

The fix is one comparison. Its strict twin makes it and refuses the same proof with `WrongEmitter`.

A FINDING WE GOT WRONG

We reported a missing function. It was there.

WHAT WE SAID	`getLogsByEventSignature` ships in the SDK's ABI but isn't deployed.
WHAT'S TRUE	**We were wrong.** Both overloads are deployed and both work — 16 of 16 functions dispatch.
WHY	`EvmV1Decoder` is a *library*. A public library function names a struct parameter canonically, so it dispatches `0x07648c7a`. Standard ABI tooling derives `0xe6c11b43` from the ABI's `type` field — and that reverts with no data, indistinguishable from a function that isn't there.
HOW IT SURFACED	Not by our testing. Our saved transcript agreed with us every time we looked. Creditcoin asked for our **reproduction** rather than our conclusion — and the reproduction is what broke.

So every finding we publish ships with the command that produces it.

An operator bonds because bonding buys credit. A proven failure record prices it.

WHO

Anyone whose transactions other people depend on — relayers, keepers, solvers, bridge and intent executors. Their counterparties can't see how reliable they are. **A scoped bond and a record nobody can edit** let them.

WHY CREDITCOIN

The record and the line belong on a chain built for credit history. **Attestcoin is how an Ethereum failure arrives as evidence without an oracle** — and the price is a pure function anyone can evaluate. A pricing rule nobody can check is an oracle.

Built: the bond, scoped coverage, the slash, the price. Not built: drawing on the line.

`ArrearsCreditLine.quote(1,000 tCTC, 500 bps, strikes)`

STRIKES	LIMIT · TCTC	PREMIUM
0	1,000.00	500 bps
1	750.00	650 bps
2	562.50	800 bps
3	421.88	950 bps
4	316.41	1,100 bps
5	237.30	1,250 bps

−25% compounding and +150 bps per strike, read back from the deployed contract. The demo operator sits at strike 5 — exactly the highlighted row — as at CC3 block 5,463,719.

WHAT WE TAKE ON TRUST

One thing, and the precompile does not verify it.

Everything else here is proven — that a transaction was included, that it failed, what it called, how much gas it burned. But **the bond lives on Creditcoin while the evidence names an Ethereum address**, and nothing in an Attestcoin proof connects those two identities.

The registry closes the gap with an EIP-191 signature, checked with `ecrecover`. Sound in practice — nobody gains by binding an address they don't control. But a compromised source-chain key means a wrongly attributable bond that no amount of proving would catch.

Named in the README's limitations and on the site — not in a code comment.

AND THE LIMITS WE CHOSE

No claimant reward.
A bounty would make the beneficiary something a paying relayer could redirect.

A careful operator can fail without consequence.
Only `gasUsed >= gasLimit` is slashable.

Settlement-time, not interception.
Broadcast to provable is 41 blocks, about eight minutes, measured.

Testnet, unaudited.
The bond is play money until it is not.

WHAT'S LIVE

Open any of it. We don't control the explorer.

CREDITCOIN CC3 TESTNET · VERIFIED ON BLOCKSCOUT

ArrearsRegistry	0x66bBEAe9…f56454
ArrearsCourt	0xdcB57306…be702f
ArrearsCreditLine	0x53853218…5f38DD
VerdictProbe	0x57b758c8…3c9e52

SITE arrears.0xo.in

SOURCE github.com/Arrears-Protocol/arrears

```
$ cd demo && npm install && npm run verify
```

both exploit halves and all seven mainnet failures, re-read live · no key, no funds, no .env

`PROTOCOL-FINDINGS.md` — six findings, each with its probe and the raw transcript. `selftest/` walks every write path in a real browser with a real wallet.

Publish the reproduction, not the result.