Onchain Escrow: A Trustless Protocol for Bilateral Settlement Without Intermediaries

Otoshi
https://onchain-escrow.com

Abstract

We present a fully decentralized smart contract protocol for trustless two-party escrow settlement on Ethereum. The protocol allows a buyer to deposit ETH or ERC-20 tokens into an immutable contract, with release controlled entirely by deterministic rules rather than human discretion. A 72-hour auto-release mechanism protects sellers from unresponsive buyers, while a 30-day dispute timeout guarantees that funds can never be permanently locked under any circumstance. The contract has no owner, no administrator, no upgrade mechanism, and no pause function. Once deployed, it operates autonomously and cannot be altered, censored, or seized by any party, including its creators.

1. Introduction

The growth of cryptocurrency-denominated commerce has exposed a fundamental coordination problem: in any bilateral transaction where delivery and payment cannot occur simultaneously, one party must extend trust to the other. The buyer who pays before receiving delivery risks losing funds if the seller does not perform. The seller who delivers before receiving payment risks providing labor or goods without compensation. This is the classical escrow problem, and it has existed for as long as commerce itself.

Traditional escrow services solve this problem by introducing a trusted third party who holds funds until both sides fulfill their obligations. Banks, legal firms, and centralized platforms have served this role for centuries. However, the introduction of a third party creates three new categories of risk that did not exist in the original bilateral relationship:

Counterparty risk. The escrow agent itself can fail. It may become insolvent, suffer a security breach, act negligently, or behave dishonestly. The parties who sought to eliminate bilateral trust have merely redirected it toward a new counterparty whose reliability they cannot independently verify.

Jurisdiction risk. The escrow agent operates within a legal jurisdiction that may freeze, seize, or redirect funds based on court orders, sanctions, or regulatory actions that are unrelated to the merits of the underlying transaction.

Censorship risk. The escrow agent retains discretionary authority to refuse service, delay settlement, or impose additional conditions not contemplated in the original agreement between the parties.

Multi-signature wallet arrangements partially address these concerns but introduce their own limitations: they require ongoing coordination between key holders, create operational complexity, and still depend on the honesty and availability of the co-signers.

This paper presents an alternative approach. Onchain Escrow replaces the trusted third party with an immutable smart contract that enforces settlement rules through code rather than human judgment. The protocol satisfies five properties simultaneously: (1) the buyer cannot lose funds without receiving delivery, (2) the seller cannot be defrauded after delivering, (3) funds cannot be permanently locked under any circumstance, (4) no third party can censor or modify the escrow, and (5) the protocol operates indefinitely without human intervention.

2. Protocol Design

The protocol implements a factory pattern in which a single contract deployment creates an unlimited number of independent escrows, each identified by a sequential integer ID. Each escrow is a self-contained state machine with four possible states: Active, Released, Cancelled, and Disputed. State transitions are governed by deterministic rules that depend only on the caller identity and the current block timestamp. No external input, oracle, or administrative action is required for any state transition.

2.1 Escrow Creation

A buyer creates an escrow by calling one of two creation functions: createEscrowETH() for native ETH deposits, or createEscrow() for ERC-20 token deposits. The buyer specifies four parameters: the seller address, the deadline timestamp, the deposit amount, and a terms hash (the keccak256 digest of the off-chain agreement between the parties).

For ERC-20 deposits, the contract measures the actual amount received using a balance-before and balance-after pattern. This accommodates tokens that implement transfer taxes or fees, where the amount received by the contract differs from the amount specified in the transfer call. The measured amount becomes the canonical deposit amount stored in the escrow record.

The deadline must be at least one hour in the future relative to the creation block timestamp. The buyer and seller addresses must be distinct. The deposit amount must be non-zero after any transfer tax deduction.

2.2 Release

The buyer may call release() at any time to transfer the deposited funds to the seller. This is the expected flow for successful transactions: the buyer deposits, the seller delivers the agreed-upon goods or services, and the buyer releases payment. The release is atomic and final. A protocol fee is deducted from the released amount and credited to the immutable fee recipient address.

The buyer may release funds before, at, or after the deadline. The deadline governs only the auto-release mechanism, not voluntary release. This design ensures that the fastest possible settlement path has no artificial delays.

2.3 Auto-Release Mechanism

If the buyer does not release after the deadline, the seller may initiate the auto-release process by calling requestRelease(). This function records the current timestamp as the release request time and begins a 72-hour grace period.

The 72-hour duration was chosen to provide adequate time for the buyer to respond across time zones and over weekends, while not imposing an unreasonable delay on the seller. During this window, the buyer has three options: release the funds voluntarily, dispute the release request, or do nothing.

If the buyer does not dispute within 72 hours, the seller calls claimRelease() to receive the funds. This mechanism converts buyer inaction into implicit approval, protecting sellers from buyers who abandon transactions without formally releasing.

2.4 Dispute Resolution

If the buyer calls dispute() within the 72-hour grace window, the escrow transitions to the Disputed state. In this state, the auto-release is blocked and both parties must resolve the dispute through one of three paths:

Voluntary release. The buyer may call release() at any time during or after a dispute, sending the full deposit to the seller.

Mutual cancellation. Both parties call mutualCancel(). When both have called, the buyer receives a full refund with no protocol fee. This provides a cooperative resolution path for disputes where both parties agree the transaction should be unwound.

Deterministic timeout. If neither party acts for 30 days after the dispute was filed, the seller may call forceResolveDispute() to claim the funds. This is the critical safety mechanism that prevents permanent fund lock.

The 30-day timeout creates a dominant strategy for honest buyers: if the seller legitimately delivered, the buyer should release promptly rather than incur a 30-day delay that ends with the same outcome. If the seller did not deliver, the buyer should propose mutual cancellation. In neither case does the buyer benefit from indefinite inaction.

The dispute window is bounded: the buyer may only file a dispute during the 72-hour grace period, not after. This prevents a buyer from waiting until the seller attempts to claim and then filing a last-minute dispute to extend the timeline indefinitely.

2.5 Mutual Cancellation

Either party may propose cancellation at any time during the Active or Disputed states by calling mutualCancel(). The function records the proposal. When both parties have proposed, the escrow transitions to Cancelled and the full deposit is returned to the buyer with no protocol fee.

Cancellation requires bilateral consent. A single party cannot unilaterally cancel an escrow, as this would allow a buyer to deposit, receive delivery, and then cancel to reclaim funds.

3. Security Architecture

The contract inherits OpenZeppelin's ReentrancyGuard, applying non-reentrant modifiers to all state-changing external functions. All ERC-20 transfers use OpenZeppelin's SafeERC20 library, which handles tokens with non-standard return values. All ETH transfers follow the Checks-Effects-Interactions pattern: state is updated before any external call is made.

Existence guards validate that the escrow ID refers to a created escrow before any operation. This prevents operations on non-existent IDs, which would otherwise interact with zero-initialized storage and potentially create phantom records.

The contract contains no selfdestruct instruction, no delegatecall, no external contract dependencies, and no oracle requirements. Its behavior is fully determined by its own bytecode and the state of the Ethereum blockchain.

4. The Case for Immutability

A widely held assumption in smart contract development is that upgrade mechanisms are necessary for fixing bugs and responding to unforeseen circumstances. We argue that for settlement infrastructure, immutability is not merely acceptable but actively preferable to upgradeability.

An upgradeable contract has a strictly larger attack surface than an immutable one. Every upgrade mechanism requires at least one privileged address with the authority to modify the contract's behavior. This address becomes a permanent vulnerability: it can be compromised through private key theft, social engineering, legal coercion, or insider malfeasance. The history of decentralized finance contains numerous examples of funds lost through compromised administrative keys.

Governance mechanisms do not eliminate this risk; they distribute it. A governance token creates a market for control, where the cost of capturing a protocol equals the cost of acquiring sufficient voting power. Time-locked governance adds delay but does not prevent a sufficiently motivated adversary from executing a malicious proposal.

An immutable contract eliminates the entire category of administrative risk. There is no key to steal, no governance to capture, no upgrade to exploit, and no emergency function to abuse. The contract's behavior is fixed at deployment and can be verified by reading its source code. Users do not need to trust the deployer, the development team, or any governance process. They need only trust the code and the Ethereum Virtual Machine.

The strongest test of a protocol's decentralization is whether it would continue to function if its creators ceased to exist. Every Onchain Escrow deployment passes this test. The contract requires no keeper bots, no oracle subscriptions, no server infrastructure, and no ongoing maintenance. It will process escrow transactions for as long as the Ethereum network produces blocks.

5. Immutability Properties

```
Owner: None
Admin functions: None
Proxy / Upgrade: None
Pause mechanism: None
Emergency withdraw: None
```

```
Grace period: 72 hours (hardcoded constant)
Dispute timeout: 30 days (hardcoded constant)
Fee rate: Hardcoded constant
Fee recipient: Set at deployment, immutable
```

6. Contract

```
Address: 0x077364eeA6099F8b7dD70b8FBA1a2569f8AD619A
Chain: Ethereum Mainnet (Chain ID: 1)
Owner: None
Compiler: Solidity ^0.8.24, Optimizer 200 runs
```

7. Conclusion

Onchain Escrow demonstrates that trustless bilateral settlement can be achieved without intermediaries, arbitrators, or governance mechanisms. The protocol replaces human judgment with deterministic timeouts and game-theoretic incentives, creating a system where honest behavior is the dominant strategy for all participants under all circumstances.

The auto-release mechanism, bounded dispute window, and 30-day timeout together guarantee a critical invariant: no escrow can lock funds permanently, regardless of participant behavior. This invariant holds without any administrative intervention, external oracle, or governance process.

The contract is permanent settlement infrastructure. It requires no team to operate, no server to maintain, and no governance to manage. It will process escrow transactions for as long as the Ethereum network produces blocks.

References

[1] EIP-20: Token Standard. Ethereum Improvement Proposals, 2015.

[2] OpenZeppelin Contracts. Security library for Solidity smart contracts.

[3] N. Szabo, "Formalizing and Securing Relationships on Public Networks," First Monday, 1997.

[4] EIP-6963: Multi Injected Provider Discovery. Ethereum Improvement Proposals, 2023.

[5] C. Detrio, "Smart Contract Security Best Practices," ConsenSys, 2019.

— Otoshi

https://onchain-escrow.com