Consensix Labs

Transparent Charitable Donation Tracking

A Smart Contract Protocol for Real-Time Fund Accountability, with a Proof-of-Concept Implementation on Ethereum

Table of Contents

Executive Summary

Charitable giving depends on trust, and that trust is frequently misplaced. Donors contribute to causes they care about but have no independent way to verify how their money is spent. Charities self-report fund allocation through annual reports and audits – processes that are infrequent, retrospective, and entirely controlled by the charity itself. When fraud or mismanagement occurs, it is typically discovered long after the funds are gone.

This paper presents a smart contract protocol that creates a transparent, independently verifiable record of the full lifecycle of charitable funds: from donation, through allocation to specific purposes, to final disbursement to recipients. A single Solidity contract manages multiple campaigns, recording every donation, every allocation decision, and every disbursement on-chain with timestamps, amounts, and cryptographic hashes linking to off-chain evidence. Anyone – donors, regulators, journalists, the general public – can query the contract and trace exactly how funds flow, in real time, without the charity's permission or cooperation.

The blockchain's role here is pure accountability. It does not replace the charity's operations or decision-making. The charity still decides how to spend the money. The blockchain records those decisions transparently, creating an audit trail that is immutable, timestamped, and publicly accessible. Off-chain evidence – invoices, purchase orders, delivery confirmations – is linked by SHA-256 hashes, keeping gas costs low while preserving verifiability.

The proof of concept implements the protocol as a single Solidity 0.8.28 contract (`DonationTracker.sol`) and demonstrates the complete campaign lifecycle using shell scripts and curl against a local Hardhat node. A "Clean Water Initiative" scenario walks through campaign creation, donations from three donors, fund allocation to two purposes, disbursement to recipients, and full audit trail queries. A second scenario demonstrates the refund mechanism for campaigns that fail to reach their funding goal.

1. Introduction

The Problem

A donor sends $100 to a disaster relief campaign. What happens next is, for all practical purposes, invisible. The charity acknowledges the donation, perhaps with a tax receipt. Months later, an annual report may summarize how the organization spent its funds across all programs. But the donor cannot trace their specific contribution to a specific outcome. They cannot verify whether the disaster relief campaign received the funds, how much was spent on supplies versus administration, or whether the supplies actually reached the affected region.

This opacity is not the exception – it is the standard operating model for charitable giving. Charities maintain their own financial records, produce their own reports, and commission their own audits. External oversight exists – organizations like Charity Navigator, Candid (formerly GuideStar), and the BBB Wise Giving Alliance evaluate charities based on financial health, governance practices, and transparency disclosures – but these evaluations are periodic, aggregate, and based largely on information the charity chooses to provide.

The consequences are well documented. High-profile charity fraud cases erode public confidence across the sector. Even absent fraud, the lack of granular transparency creates a principal-agent problem: donors (the principals) cannot effectively monitor how charities (their agents) use their funds. This information asymmetry suppresses giving. Surveys consistently find that concerns about how donations are used are the primary barrier to charitable giving, ahead of financial constraints.

The problem exists at every scale. Individual donors giving small amounts have no practical way to verify fund usage. Institutional donors and grant-making organizations funding programs across multiple countries face the same challenge magnified – they rely on self-reported metrics from recipients, with verification requiring expensive in-person audits. Government agencies distributing disaster relief funds through charitable intermediaries face accountability gaps that are difficult to close retroactively.

The root cause is structural: donation records, allocation decisions, and disbursement evidence live in the charity's own systems. There is no common, independently verifiable record that a third party can audit without the charity's cooperation.

The Opportunity

A smart contract on a public blockchain can serve as a transparent, tamper-proof ledger for the full lifecycle of charitable funds. Every transaction – donation, allocation, disbursement – is publicly visible, timestamped, and irreversible. The contract holds donated funds in escrow until the charity explicitly allocates and disburses them, creating a clear separation between receiving funds and spending them. Each step is recorded with enough detail for any observer to reconstruct the complete flow of funds from donation to final disbursement.

The key insight is that the blockchain does not need to replace the charity's operations. The charity retains full control over how funds are spent. What changes is that every spending decision becomes a public, permanent record. An allocation declares intent (this money will be used for medical supplies in region X). A disbursement records execution (this amount was sent to this recipient for this purpose, with this evidence). The blockchain creates an independent audit trail alongside the charity's existing systems.

This is blockchain as an accountability layer – the same principle that underpins other practical blockchain applications. The on-chain data is deliberately minimal: amounts, addresses, timestamps, and hashes. All descriptive content (campaign descriptions, allocation purposes, evidence documents) lives off-chain, linked by SHA-256 hashes. This keeps gas costs low while preserving verifiability: anyone can check that an off-chain document matches its on-chain hash.

Scope and Approach

This paper presents a protocol for transparent charitable donation tracking and a working proof of concept that implements it. The PoC consists of a single Solidity smart contract and shell scripts that demonstrate the complete lifecycle using a local Ethereum node.

The approach is intentionally minimal. There is no web interface, no backend service, and no off-chain storage infrastructure. The demonstration uses bash scripts with curl and jq to interact with the contract via JSON-RPC. This keeps the focus on the protocol mechanics – the smart contract design, the data model, and the audit trail – without the complexity of application-level concerns.

Several features are explicitly out of scope. The contract handles ETH donations only; stablecoin support and fiat on-ramp integration are discussed as future directions. Donor governance (voting on fund allocation) is not implemented – the charity retains full operational control. Milestone-based conditional release (where funds are released only when verified outcomes are achieved) is a natural extension discussed in Section 9 but not implemented here. The contract is a demonstration of a concept, not a production system.

2. Background

Charitable Giving Landscape

Global charitable giving represents a significant economic sector. In the United States alone, individual, corporate, and foundation giving exceeds $500 billion annually. The sector supports millions of organizations ranging from small local nonprofits to international relief agencies operating across dozens of countries.

The accountability infrastructure for this sector has evolved primarily around annual reporting. In the United States, tax-exempt organizations file IRS Form 990, which discloses revenue, expenses, executive compensation, and program activities. These filings are public, and services like Candid aggregate and organize them. Charity Navigator rates over 230,000 organizations using a multi-dimensional scoring system covering financial health, accountability, and impact. The BBB Wise Giving Alliance evaluates charities against 20 standards covering governance, effectiveness, finances, and solicitations.

These mechanisms provide valuable macro-level transparency. A donor can determine whether a charity spends a reasonable percentage of its revenue on programs versus administration, whether its governance practices meet accepted standards, and whether its financial position is stable. What these mechanisms cannot provide is per-campaign, per-donation, real-time transparency. They operate on annual cycles, evaluate organizations rather than specific campaigns, and rely on self-reported data.

Existing Blockchain Approaches

Several projects have applied blockchain technology to charitable giving. The following table summarizes the landscape.

Approach	Model	Transparency Scope	Fund Custody	Status
The Giving Block	Crypto donation platform	Donation receipt	Charity-controlled	Active (Shift4 subsidiary)
Gitcoin Grants	Quadratic funding	Funding allocation	Smart contract escrow	Active
Alice.si	Outcome-based funding	Impact verification	Smart contract escrow	Active (pivoting to impact finance)
Giveth	DAO-governed giving	Donation tracking	Charity-controlled	Active
GiveTrack (BitGive)	Donation tracking	Donation and project tracking	Charity-controlled	Active (limited scale)
UNICEF CryptoFund	Crypto venture fund	Investment tracking	UNICEF-controlled	Active (expanding to stablecoins)
GiveDirectly	Direct cash transfers	Transfer verification	Organization-controlled	Active (not blockchain-based)

The Giving Block is the leading platform for cryptocurrency donations to nonprofits. Acquired by Shift4 in 2022, it has facilitated over $300 million in crypto donations since 2018 and supports over 100 cryptocurrencies. Its 2026 annual report highlights crypto donors consistently making larger-than-average gifts. However, The Giving Block is a donation platform, not a transparency protocol. Once a donation reaches the charity, the platform provides no visibility into how those funds are allocated or spent.

Gitcoin Grants pioneered quadratic funding for public goods in the Ethereum ecosystem. The model amplifies small donations through a matching pool, where matching amounts are proportional to the number of contributors rather than the total contributed. This is a funding mechanism innovation, not a fund-tracking mechanism. Gitcoin's protocol determines how funds are allocated to projects but does not track how projects subsequently spend those funds.

Alice.si is the closest conceptual predecessor to the protocol presented here. Built on Ethereum, Alice uses smart contracts to hold donations in escrow and releases funds to charities only when verified outcomes are achieved. A dedicated validator confirms goal achievement before payment. Alice ran a successful pilot with St Mungo's in London helping rough sleepers find housing. The project has evolved toward broader impact finance, including decentralized social impact bonds and blockchain-based beneficiary accounts. Alice's outcome-based model is more restrictive than the protocol presented here (which allows charities to allocate freely, recording decisions transparently rather than gating them) but demonstrates that smart contract escrow for charitable funds is technically viable and operationally practical.

Giveth is an open-source platform using decentralized autonomous organizations (DAOs) to manage charitable funds. It introduced a "Causes" feature combining AI-driven project curation with blockchain transparency. Giveth focuses on community governance of funding decisions rather than per-campaign fund tracking.

GiveTrack, built by the BitGive Foundation, provides real-time tracking of Bitcoin donations through to project outcomes. It demonstrates the concept of donor-visible tracking but operates at limited scale and is restricted to Bitcoin.

UNICEF CryptoFund is a notable institutional example. Launched in 2019 as the first United Nations vehicle to hold and disburse cryptocurrency, it has invested over $4 million in BTC and ETH into startups in developing economies. In early 2026, UNICEF announced the addition of USDC stablecoins to the fund. The CryptoFund publishes all transactions on public blockchains, providing inherent transparency through the ledger itself. However, this transparency covers investment disbursements, not the downstream impact of those investments.

GiveDirectly is not blockchain-based but represents the gold standard for donation transparency. The organization delivers unconditional cash transfers directly to people in extreme poverty, with published research and a real-time GDLive newsfeed where recipients share unedited updates. GiveDirectly has disbursed approximately $950 million to date and consistently achieves 85-90% of donations reaching recipients. The organization demonstrates that radical transparency is achievable and valued by donors, even without blockchain technology.

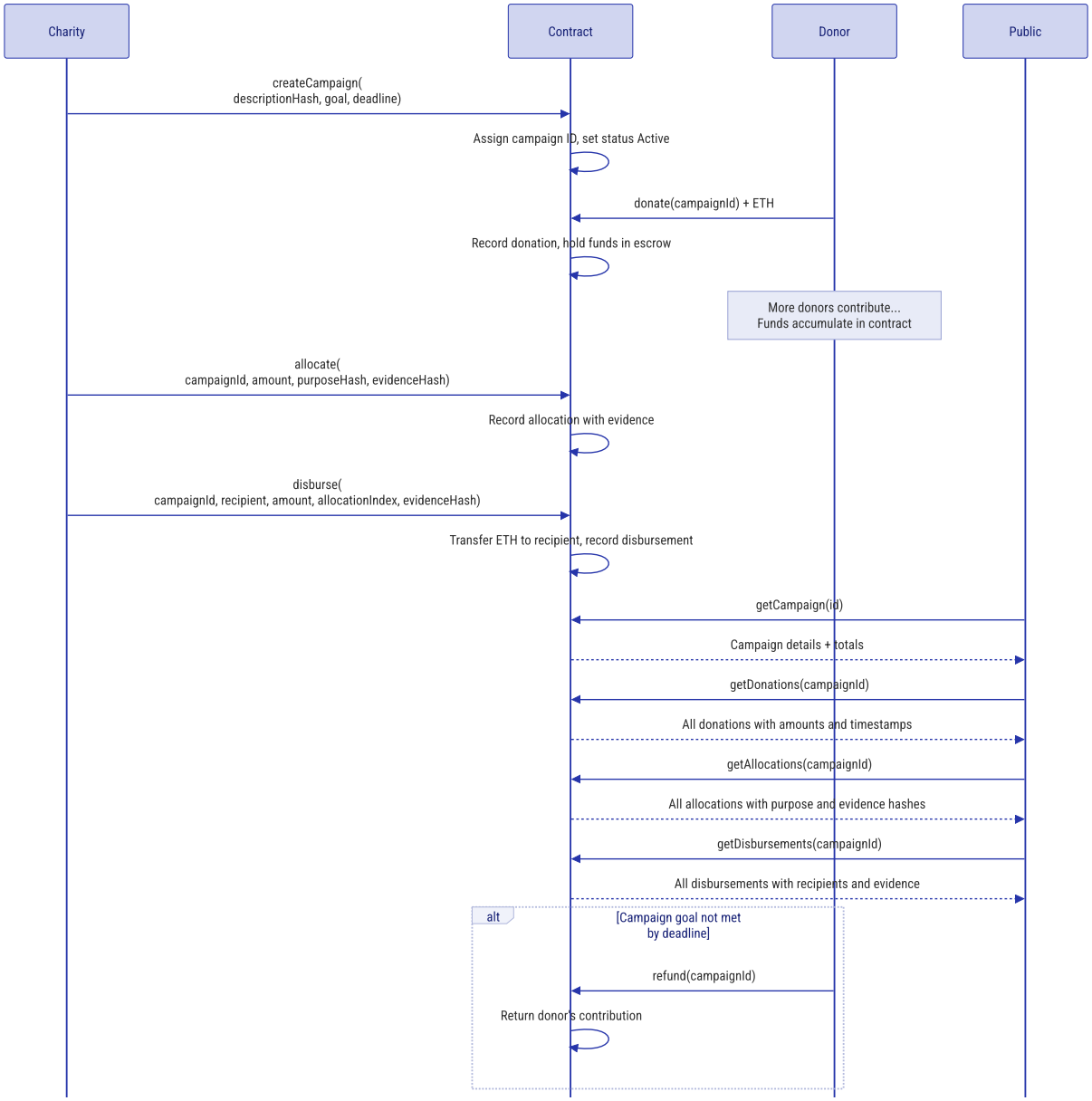
Traditional Accountability Mechanisms

The traditional charity accountability ecosystem operates on three levels. **Self-reporting** through annual reports, IRS Form 990 filings, and charity-specific publications provides the base layer. **Third-party evaluation** by Charity Navigator (star ratings), Candid (transparency seals), and BBB Wise Giving Alliance (accreditation) provides independent assessment. **Auditing** by professional accounting firms provides the highest level of assurance but is expensive, infrequent, and backward-looking.

The protocol presented here is complementary to these mechanisms, not a replacement. Annual ratings evaluate organizational health and governance – important factors that a per-campaign smart contract cannot assess. Professional audits verify accounting practices and internal controls. What the protocol adds is a layer that these mechanisms cannot provide: real-time, per-campaign, per-donation transparency that operates continuously rather than annually, and is independently verifiable without the charity's cooperation.

3. The Protocol

How It Works



The protocol has four phases:

1. **Campaign creation.** A charity creates a campaign by providing a description hash (linking to off-chain campaign details), a funding goal in ETH, and a deadline. The contract assigns a campaign ID and marks the campaign as active. The charity's Ethereum address becomes the campaign creator, controlling allocation and disbursement operations.

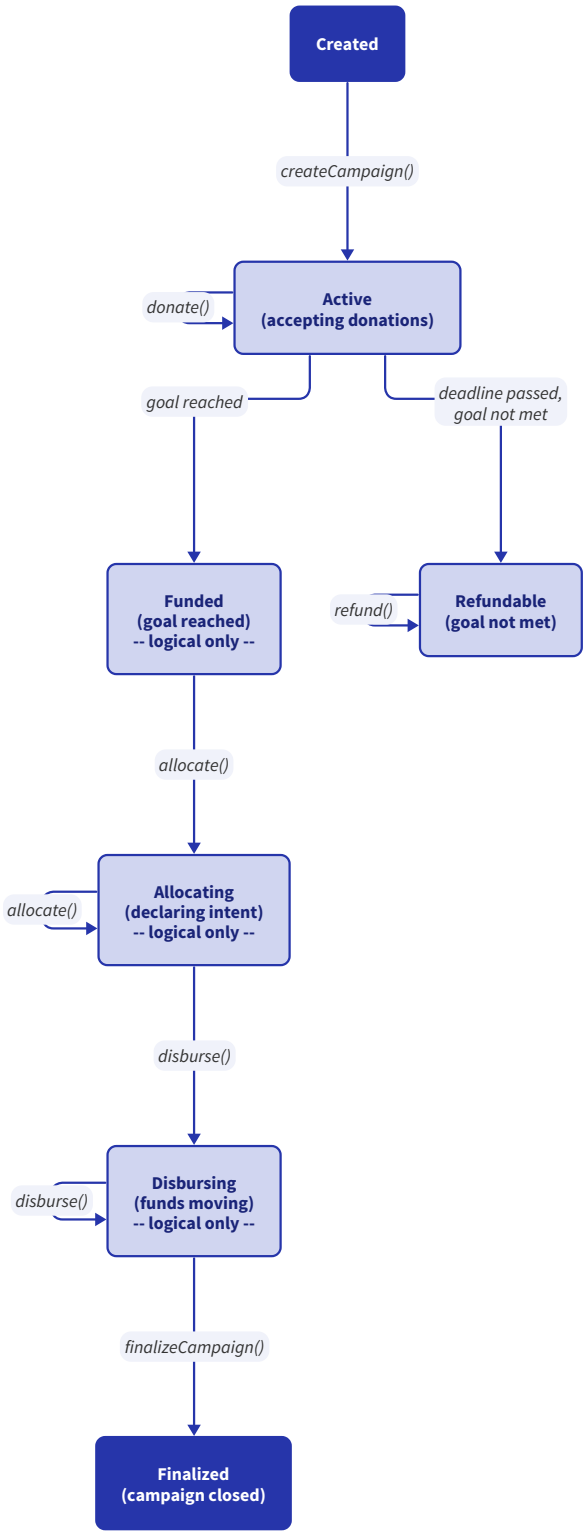
2. **Donations.** Anyone can donate ETH to an active campaign. Each donation is recorded on-chain with the donor's address, the amount, a timestamp, and an optional message hash (linking to off-chain donor intent or dedication). The funds are held in the smart contract, not transferred to the charity. This escrow model is central to the protocol's accountability guarantee – the charity must explicitly allocate and disburse funds through recorded on-chain actions.

3. **Allocation.** The charity allocates portions of the campaign funds to specific purposes. Each allocation records the amount, a purpose description hash (linking to off-chain details such as "water purification equipment for region X"), and an evidence hash (linking to supporting documentation – an invoice, a purchase order, a vendor agreement). Allocations are public commitments: they declare intent before money moves.

4. **Disbursement.** The charity disburses allocated funds to recipient addresses. Each disbursement records the recipient, the amount, the allocation it fulfills, and an evidence hash (linking to proof of delivery – a receipt, a photo, a delivery confirmation). The actual ETH transfer happens at this point. The funds move from the contract to the recipient, and the disbursement is permanently recorded.

Additional mechanics provide safety and lifecycle management. A **refund window** protects donors if a campaign fails to reach its goal by its deadline – donors can reclaim their contributions. **Campaign finalization** allows the charity to close a campaign to further donations once the goal is met or the deadline passes. **Query functions** allow anyone to read campaign details, donations, allocations, disbursements, and balance information at zero gas cost.

Campaign Lifecycle



The state machine reflects a deliberate simplification. In practice, the contract allows allocation and disbursement at any time after a campaign receives donations – the “funded” and “allocating” states are logical rather than enforced. The contract enforces two hard constraints: only the campaign creator can allocate and disburse, and donors can only refund after the deadline has passed and the goal was not met.

Trust Model

The protocol uses an **open charity model**: any Ethereum address can create a campaign. There is no approval process, no registry of approved charities, and no gatekeeper. This mirrors how charitable giving works in practice – anyone can solicit donations, and the donor decides whom to trust.

The blockchain provides **transparency, not truth**. The contract records every transaction accurately, but it does not verify that off-chain evidence is genuine. An evidence hash proves that the charity committed to a specific document at a specific time, but not that the document is truthful. A photo of delivered supplies could be staged. An invoice could be inflated. The blockchain provides accountability (the charity publicly committed to this evidence and it cannot be changed after the fact), not truth verification.

This is an important distinction. The protocol makes it significantly harder to hide misuse of funds – every allocation and disbursement is a permanent public record – but it does not make misuse impossible. The value lies in the audit trail: if questions arise, every decision is documented, timestamped, and immutable.

Advantages

Real-time transparency. Fund status is visible at any moment, not just at annual reporting intervals. A donor can check how much has been raised, how much has been allocated, and how much has been disbursed – today, not after the next audit cycle.

Independent verification. Anyone with Ethereum access can query the contract directly. No API key, no platform account, no permission from the charity. The data is on a public blockchain and freely accessible.

Per-campaign granularity. Unlike annual reports that aggregate spending across all programs, this protocol tracks funds at the campaign level. A donor to a specific disaster relief campaign can see exactly how that campaign's funds were used.

Escrow by default. Donated funds are held in the smart contract until explicitly disbursed. The charity must take a recorded on-chain action to move funds, creating an accountability checkpoint at every step.

Evidence linking. Every allocation and disbursement includes a hash linking to off-chain evidence. While the evidence itself is not verified by the contract, the hash creates a permanent, tamper-proof commitment to specific documentation.

Limitations

No truth verification. The protocol records what the charity claims to have done, not what actually happened. Off-chain evidence could be fabricated. The blockchain provides a transparent record of claims, not a guarantee of their accuracy. Addressing this gap requires external verification mechanisms – human auditors, AI-assisted evidence analysis, or oracle-based milestone verification – which are discussed as future directions.

ETH only. The proof of concept handles ETH donations only. Real-world charitable giving overwhelmingly uses fiat currency. Stablecoin support would bridge this gap significantly, and fiat on-ramp integration is a longer-term requirement. Both are discussed in Section 9.

No donor governance. Donors have no say in how funds are allocated – the charity retains full operational control. This is a deliberate design choice (charities have operational expertise that donors typically lack), but some use cases might benefit from donor input on allocation decisions.

Gas costs. Every on-chain operation costs gas. While individual operations are modest, a campaign with thousands of donors would incur significant cumulative costs. Layer 2 solutions and alternative blockchain platforms (discussed in Section 8) can address this.

No privacy for donors. Donor addresses and amounts are publicly visible on-chain. While Ethereum addresses are pseudonymous, this may not provide sufficient privacy for all donors. Future work on selective disclosure and zero-knowledge proofs could address this.

Allocation index is a claim, not a constraint. When disbursing, the charity specifies which allocation the disbursement fulfills, but the contract does not enforce that the disbursement purpose matches the allocation's stated purpose. A charity could disburse funds tagged to "water purification equipment" toward an unrelated recipient. This is consistent with the protocol's "transparency, not truth" philosophy – the on-chain record documents the charity's claim, and any mismatch between allocation purpose and disbursement recipient is visible to auditors. But readers should understand that `allocationIndex` is a labeling mechanism, not an enforcement mechanism.

Finalization closes donations, not operations. Calling `finalizeCampaign()` prevents further donations but does not block allocation or disbursement. This is intentional – a charity needs to continue disbursing after donations close – but means the "Finalized" state is narrower than it might initially suggest.

What This Is NOT

This protocol is **not a replacement for charity regulation or auditing**. Regulatory oversight, professional audits, and organizational governance remain essential. The protocol adds a layer of real-time transparency that complements these existing mechanisms.

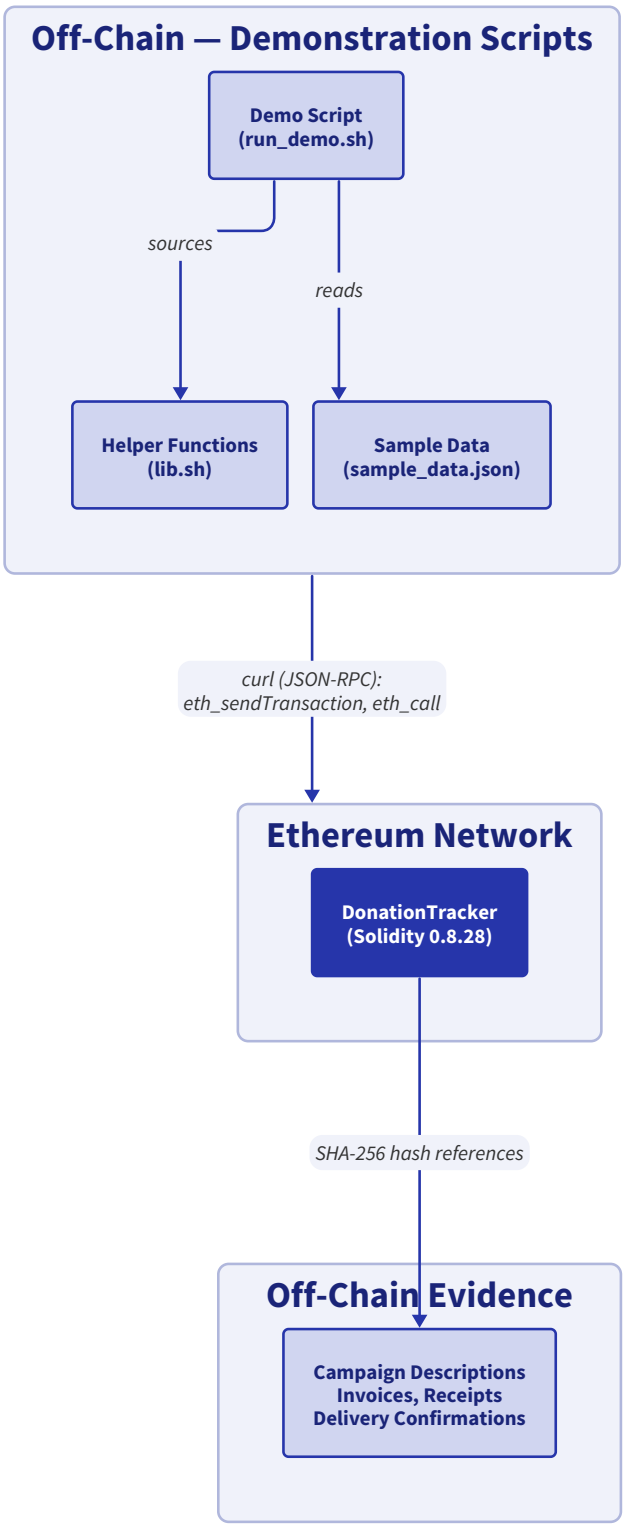
This protocol **does not verify that off-chain evidence is genuine**. An evidence hash proves that the charity committed to a specific document, but not that the document is truthful. The blockchain provides accountability, not truth verification.

This protocol **does not handle fiat currency donations**. The PoC works with ETH only. Integration with fiat on-ramps and stablecoins is a future direction.

This protocol **does not implement governance or voting**. Donors do not vote on how funds are spent. The charity retains full operational control. The blockchain only records their decisions transparently.

4. Architecture

System Overview



The system has three layers. The **on-chain layer** is a single Solidity smart contract that stores campaign state, donations, allocations, and disbursements. It holds donated ETH in escrow and enforces access control for fund management operations. The **off-chain demonstration layer** consists of shell scripts that deploy the contract and exercise the complete campaign lifecycle using curl against the Ethereum JSON-RPC interface. The **off-chain evidence layer** represents the external documents (campaign descriptions, invoices, receipts, delivery confirmations) that are linked to on-chain records via SHA-256 hashes.

The architecture reflects a specific philosophy: the blockchain stores the minimum needed for accountability (amounts, addresses, timestamps, hashes), and everything else stays off-chain. This keeps gas costs low while preserving the audit trail's integrity.

Smart Contract Design

The contract manages three core data structures:

Campaigns are created by charities and define the fundraising parameters. Each campaign has a creator address, a description hash, a funding goal, a deadline, running totals for donated, allocated, disbursed, and refunded amounts, and a status. The `totalDonated` field is monotonically increasing – it records the cumulative total of all donations and is never decremented. Refunds are tracked separately in `totalRefunded`, so the campaign balance is always `totalDonated - totalDisbursed - totalRefunded`. The creator address serves as the access control mechanism – only the creator can allocate and disburse campaign funds.

Donations are recorded per campaign. Each donation stores the donor's address, the amount, a timestamp, and an optional message hash. Donations are append-only – once recorded, they cannot be modified or deleted.

Allocations declare the charity's intent to use funds for a specific purpose. Each allocation stores an amount, a purpose hash (linking to off-chain details), an evidence hash (linking to supporting documentation), and a timestamp. Allocations do not move funds – they are commitments that precede disbursements.

Disbursements execute the actual fund transfer. Each disbursement stores the recipient address, the amount, the index of the allocation it fulfills, an evidence hash (linking to proof of delivery or completion), and a timestamp. The disbursement triggers an ETH transfer from the contract to the recipient.

On-Chain vs Off-Chain Data

On-Chain (per record)	Off-Chain
Amounts (ETH)	Campaign descriptions
Addresses (charity, donors, recipients)	Allocation purpose details
Timestamps	Invoices and purchase orders
SHA-256 hashes	Delivery confirmations
Status flags	Photographs and reports
Allocation and disbursement indices	Donor messages and dedications

The on-chain footprint is deliberately minimal. A campaign description might be several pages of text with images – storing it on-chain would cost thousands of dollars in gas. Instead, the description is stored anywhere convenient (IPFS, a website, a PDF) and its SHA-256 hash is recorded on-chain. Anyone can verify that

a specific off-chain document matches the hash recorded at campaign creation time.

5. Smart Contract Implementation

DonationTracker.sol

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.28;

/// @title DonationTracker
/// @notice Transparent on-chain ledger for the full lifecycle of charitable
///         campaign funds: donation, allocation, and disbursement.
///         Funds are held in escrow until explicitly disbursed by the
///         campaign creator, and every operation is publicly queryable.
contract DonationTracker {

    // -- Campaign status ----------------------------------------------------

    enum CampaignStatus {
        Active,     // 0 - accepting donations
        Finalized   // 1 - closed to new donations (allocation/disbursement may continue)
    }

    // -- Data structures ----------------------------------------------------

    struct Campaign {
        address creator;           // charity address that controls the campaign
        bytes32 descriptionHash;   // SHA-256 hash of off-chain campaign details
        uint256 goalAmount;        // funding target in wei
        uint256 deadline;          // unix timestamp after which donations close and refunds open
        uint256 totalDonated;      // running total of all donations (monotonically increasing)
        uint256 totalAllocated;    // running total of all allocations
        uint256 totalDisbursed;    // running total of all disbursements
        uint256 totalRefunded;     // running total of all refunds
        CampaignStatus status;
        bool exists;               // guard against default-value reads
    }

    struct Donation {
        address donor;
        uint256 amount;
        uint256 timestamp;
        bytes32 messageHash;       // optional hash of off-chain donor message
    }

    struct Allocation {
        uint256 amount;
        bytes32 purposeHash;       // hash of off-chain purpose description
        bytes32 evidenceHash;      // hash of supporting documentation
        uint256 timestamp;
    }

    struct Disbursement {
        address recipient;
        uint256 amount;
```

```
        uint256 allocationIndex;   // which allocation this fulfills (claim, not constraint)
        bytes32 evidenceHash;      // hash of proof-of-delivery documentation
        uint256 timestamp;
    }

    // -- Storage ------------------------------------------------------------

    uint256 private nextCampaignId;

    mapping(uint256 => Campaign) private campaigns;
    mapping(uint256 => Donation[]) private donations;
    mapping(uint256 => Allocation[]) private allocations;
    mapping(uint256 => Disbursement[]) private disbursements;

    // Track each donor's contribution per campaign for refund calculation
    mapping(uint256 => mapping(address => uint256)) private donorContributions;
    // Track whether a donor has already claimed a refund
    mapping(uint256 => mapping(address => bool)) private refundClaimed;

    // -- Events -------------------------------------------------------------

    event CampaignCreated(
        uint256 indexed campaignId,
        address indexed creator,
        bytes32 descriptionHash,
        uint256 goalAmount,
        uint256 deadline
    );

    event DonationReceived(
        uint256 indexed campaignId,
        address indexed donor,
        uint256 amount,
        bytes32 messageHash,
        uint256 timestamp
    );

    event FundsAllocated(
        uint256 indexed campaignId,
        uint256 amount,
        bytes32 purposeHash,
        bytes32 evidenceHash,
        uint256 timestamp
    );

    event FundsDisbursed(
        uint256 indexed campaignId,
        address indexed recipient,
        uint256 amount,
        uint256 allocationIndex,
        bytes32 evidenceHash,
        uint256 timestamp
    );

    event CampaignFinalized(
        uint256 indexed campaignId,
```

```
        uint256 timestamp
    );

    event DonationRefunded(
        uint256 indexed campaignId,
        address indexed donor,
        uint256 amount,
        uint256 timestamp
    );

    // -- Custom errors ----------------------------------------------------------

    error CampaignNotFound(uint256 campaignId);
    error CampaignNotActive(uint256 campaignId);
    error CampaignAlreadyFinalized(uint256 campaignId);
    error DonationWindowClosed(uint256 campaignId);
    error NotCampaignCreator(uint256 campaignId);
    error DeadlineInPast();
    error GoalMustBePositive();
    error DonationMustBePositive();
    error AllocationExceedsDonated(uint256 requested, uint256 available);
    error DisbursementExceedsAllocated(uint256 requested, uint256 available);
    error InvalidAllocationIndex(uint256 index, uint256 length);
    error InvalidRecipient();
    error RefundNotAvailable(uint256 campaignId);
    error NothingToRefund(uint256 campaignId, address donor);
    error RefundAlreadyClaimed(uint256 campaignId, address donor);
    error TransferFailed();

    // -- Campaign management ----------------------------------------------------

    /// @notice Create a new fundraising campaign.
    /// @param descriptionHash SHA-256 hash of off-chain campaign details
    /// @param goalAmount Funding target in wei
    /// @param deadline Unix timestamp after which donations close and refunds open
    /// @return campaignId The newly created campaign's identifier
    function createCampaign(
        bytes32 descriptionHash,
        uint256 goalAmount,
        uint256 deadline
    ) external returns (uint256 campaignId) {
        if (goalAmount == 0) revert GoalMustBePositive();
        if (deadline <= block.timestamp) revert DeadlineInPast();

        campaignId = nextCampaignId++;

        campaigns[campaignId] = Campaign({
            creator: msg.sender,
            descriptionHash: descriptionHash,
            goalAmount: goalAmount,
            deadline: deadline,
            totalDonated: 0,
            totalAllocated: 0,
            totalDisbursed: 0,
            totalRefunded: 0,
            status: CampaignStatus.Active,
```

```
            exists: true
        });

        emit CampaignCreated(
            campaignId, msg.sender, descriptionHash, goalAmount, deadline
        );
    }

    /// @notice Donate ETH to an active campaign before its deadline.
    /// @param campaignId The campaign to donate to
    /// @param messageHash Optional hash of off-chain donor message (zero if none)
    function donate(
        uint256 campaignId,
        bytes32 messageHash
    ) external payable {
        Campaign storage c = campaigns[campaignId];
        if (!c.exists) revert CampaignNotFound(campaignId);
        if (c.status != CampaignStatus.Active) revert CampaignNotActive(campaignId);
        if (block.timestamp > c.deadline) revert DonationWindowClosed(campaignId);
        if (msg.value == 0) revert DonationMustBePositive();

        c.totalDonated += msg.value;
        donorContributions[campaignId][msg.sender] += msg.value;

        donations[campaignId].push(Donation({
            donor: msg.sender,
            amount: msg.value,
            timestamp: block.timestamp,
            messageHash: messageHash
        }));

        emit DonationReceived(
            campaignId, msg.sender, msg.value, messageHash, block.timestamp
        );
    }

    /// @notice Allocate campaign funds to a specific purpose.
    ///         Does not move funds -- declares intent with evidence.
    /// @param campaignId The campaign to allocate from
    /// @param amount Amount in wei to allocate
    /// @param purposeHash SHA-256 hash of off-chain purpose description
    /// @param evidenceHash SHA-256 hash of supporting documentation
    function allocate(
        uint256 campaignId,
        uint256 amount,
        bytes32 purposeHash,
        bytes32 evidenceHash
    ) external {
        Campaign storage c = campaigns[campaignId];
        if (!c.exists) revert CampaignNotFound(campaignId);
        if (msg.sender != c.creator) revert NotCampaignCreator(campaignId);

        // Available = donated minus already allocated, minus refunded
        uint256 available = c.totalDonated - c.totalAllocated - c.totalRefunded;
        if (amount > available) {
            revert AllocationExceedsDonated(amount, available);
```

```
        }

        c.totalAllocated += amount;

        allocations[campaignId].push(Allocation({
            amount: amount,
            purposeHash: purposeHash,
            evidenceHash: evidenceHash,
            timestamp: block.timestamp
        }));

        emit FundsAllocated(
            campaignId, amount, purposeHash, evidenceHash, block.timestamp
        );
    }

    /// @notice Disburse allocated funds to a recipient.
    ///         Triggers the actual ETH transfer and records the disbursement.
    /// @param campaignId The campaign to disburse from
    /// @param recipient Address receiving the funds
    /// @param amount Amount in wei to disburse
    /// @param allocationIndex Which allocation this disbursement fulfills (claim, not constraint)
    /// @param evidenceHash SHA-256 hash of proof-of-delivery documentation
    function disburse(
        uint256 campaignId,
        address payable recipient,
        uint256 amount,
        uint256 allocationIndex,
        bytes32 evidenceHash
    ) external {
        Campaign storage c = campaigns[campaignId];
        if (!c.exists) revert CampaignNotFound(campaignId);
        if (msg.sender != c.creator) revert NotCampaignCreator(campaignId);
        if (recipient == address(0)) revert InvalidRecipient();

        if (allocationIndex >= allocations[campaignId].length) {
            revert InvalidAllocationIndex(
                allocationIndex, allocations[campaignId].length
            );
        }

        // Cannot disburse more than what has been allocated minus prior disbursements
        uint256 available = c.totalAllocated - c.totalDisbursed;
        if (amount > available) {
            revert DisbursementExceedsAllocated(amount, available);
        }

        c.totalDisbursed += amount;

        disbursements[campaignId].push(Disbursement({
            recipient: recipient,
            amount: amount,
            allocationIndex: allocationIndex,
            evidenceHash: evidenceHash,
            timestamp: block.timestamp
        }));
```

```
        // Emit before external call (CEI pattern)
        emit FundsDisbursed(
            campaignId, recipient, amount, allocationIndex,
            evidenceHash, block.timestamp
        );

        // Transfer ETH to the recipient
        (bool success, ) = recipient.call{value: amount}("");
        if (!success) revert TransferFailed();
    }

    /// @notice Finalize a campaign, preventing further donations.
    ///         Allocation and disbursement may continue after finalization.
    /// @param campaignId The campaign to finalize
    function finalizeCampaign(uint256 campaignId) external {
        Campaign storage c = campaigns[campaignId];
        if (!c.exists) revert CampaignNotFound(campaignId);
        if (msg.sender != c.creator) revert NotCampaignCreator(campaignId);
        if (c.status == CampaignStatus.Finalized) {
            revert CampaignAlreadyFinalized(campaignId);
        }

        c.status = CampaignStatus.Finalized;

        emit CampaignFinalized(campaignId, block.timestamp);
    }

    /// @notice Reclaim a donation if the campaign missed its goal.
    ///         Available only after the deadline, and only if the goal was not met.
    /// @param campaignId The campaign to request a refund from
    function refund(uint256 campaignId) external {
        Campaign storage c = campaigns[campaignId];
        if (!c.exists) revert CampaignNotFound(campaignId);

        // Refund requires: deadline passed AND goal not met
        if (block.timestamp <= c.deadline || c.totalDonated >= c.goalAmount) {
            revert RefundNotAvailable(campaignId);
        }

        if (refundClaimed[campaignId][msg.sender]) {
            revert RefundAlreadyClaimed(campaignId, msg.sender);
        }

        uint256 contributed = donorContributions[campaignId][msg.sender];
        if (contributed == 0) {
            revert NothingToRefund(campaignId, msg.sender);
        }

        refundClaimed[campaignId][msg.sender] = true;
        c.totalRefunded += contributed;

        // Emit before external call (CEI pattern)
        emit DonationRefunded(
            campaignId, msg.sender, contributed, block.timestamp
        );
```

```
        (bool success, ) = msg.sender.call{value: contributed}("");
        if (!success) revert TransferFailed();
    }

    // -- Query functions (all view, zero gas when called off-chain) -----------

    /// @notice Get campaign details.
    function getCampaign(uint256 campaignId) external view returns (
        address creator,
        bytes32 descriptionHash,
        uint256 goalAmount,
        uint256 deadline,
        uint256 totalDonated,
        uint256 totalAllocated,
        uint256 totalDisbursed,
        uint256 totalRefunded,
        CampaignStatus status,
        bool exists
    ) {
        Campaign storage c = campaigns[campaignId];
        return (
            c.creator, c.descriptionHash, c.goalAmount, c.deadline,
            c.totalDonated, c.totalAllocated, c.totalDisbursed,
            c.totalRefunded, c.status, c.exists
        );
    }

    /// @notice Get all donations for a campaign.
    function getDonations(uint256 campaignId)
        external view returns (Donation[] memory)
    {
        return donations[campaignId];
    }

    /// @notice Get all allocations for a campaign.
    function getAllocations(uint256 campaignId)
        external view returns (Allocation[] memory)
    {
        return allocations[campaignId];
    }

    /// @notice Get all disbursements for a campaign.
    function getDisbursements(uint256 campaignId)
        external view returns (Disbursement[] memory)
    {
        return disbursements[campaignId];
    }

    /// @notice Get the ETH balance held in escrow for a specific campaign.
    ///         Computed as totalDonated - totalDisbursed - totalRefunded.
    function getCampaignBalance(uint256 campaignId)
        external view returns (uint256)
    {
        Campaign storage c = campaigns[campaignId];
        if (!c.exists) revert CampaignNotFound(campaignId);
```

```
        return c.totalDonated - c.totalDisbursed - c.totalRefunded;
    }
}
```

The contract is under 400 lines (including comments and whitespace) and intentionally straightforward. It uses custom errors instead of `require` strings for gas efficiency. Events are emitted for every state change, enabling off-chain indexing and real-time monitoring. There are no proxy patterns, no upgradability mechanisms, and no `selfdestruct` – the contract is designed to be simple enough to audit by reading the source code.

The access control model is minimal: only the campaign creator can allocate, disburse, and finalize. Anyone can donate. Anyone can query. Refunds are available to donors only when the deadline has passed and the goal was not met. This keeps the security surface small.

The refund mechanism uses a pull pattern (donors call `refund()` to withdraw) rather than a push pattern (the contract sends refunds automatically). This avoids the well-known issues with push-based payment patterns in Solidity, where a failing recipient can block the entire operation.

6. Running the Demonstration

Prerequisites

The demonstration requires bash, curl, jq, python3, Docker, and the Ethereum Local Testing Starter Pack. The starter pack provides a local Hardhat node and the `eth.sh` script used by the demo to create funded accounts.

Option A: Using the Starter Pack (Local)

Step 1: Start the Ethereum network.

```
cd /path/to/Ethereum-Local-Testing-Starter-Pack
./eth.sh start
```

This starts a Hardhat node at `localhost:41545`. The demo script will create its own funded accounts using the starter pack's `eth.sh create-account` command.

Step 2: Run the demonstration.

```
cd transparent-charitable-donations
cp .env.example .env
# .env defaults work with the starter pack
./run_demo.sh
```

The script sources `demo/lib.sh` for helper functions, reads campaign descriptions and evidence text from `demo/sample_data.json` (computing SHA-256 hashes at runtime), and performs all operations via `curl` against the Hardhat node's JSON-RPC endpoint.

Option B: Using a Public Testnet (Sepolia)

For readers who want to try on a public testnet without the starter pack:

1. Set `RPC_URL` in `.env` to a Sepolia endpoint (e.g., from Infura or Alchemy).
2. Fund test accounts with Sepolia ETH from a faucet.
3. Deploy the contract using Foundry: `forge create --rpc-url $RPC_URL --private-key $KEY contracts/DonationTracker.sol:DonationTracker`.
4. Interact using `cast`: `cast send --rpc-url $RPC_URL --private-key $KEY $CONTRACT "donate(uint256,bytes32)" 0 0x00...00 --value 1ether`.

The demo scripts are written for Option A (unlocked accounts, `eth_sendTransaction`). Adapting them to Option B requires replacing `eth_sendTransaction` calls with signed transactions, but the contract and its interface are identical.

The Demo Scenario

The demonstration walks through two complete scenarios.

Scenario 1: Clean Water Initiative. A charity creates a campaign, three donors contribute, the charity allocates funds to two purposes (water purification equipment and distribution logistics), disburses to two recipients, and the full audit trail is queried.

Scenario 2: Failed Campaign with Refund. A second campaign is created with a high goal and a short deadline. A single donor contributes, but the goal is not met. The blockchain's time is advanced past the deadline using Hardhat's `evm_increaseTime` and `evm_mine` RPC methods, and the donor reclaims their contribution.

The demonstration computes SHA-256 hashes at runtime from the campaign descriptions, allocation purposes, and evidence text stored in `demo/sample_data.json`. In a production system, these hashes would be computed from actual documents at the time of each operation. The sample data file contains realistic but fictional content that represents the kind of evidence a real campaign would reference.

ABI Encoding

Ethereum contract calls require ABI-encoded calldata. The `lib.sh` helper file provides functions that construct the encoded hex strings from parameters. For example, the `createCampaign` function selector is the first 4 bytes of the Keccak-256 hash of its signature `createCampaign(bytes32,uint256,uint256)`:

```
Selector: 0x1bde12ef
```

The arguments are encoded as 32-byte, zero-padded hex values. A complete `createCampaign` call with a description hash, a goal of 5 ETH (`0x4563918244f40000` in wei), and a deadline timestamp looks like:

```
0x1bde12ef
  <32 bytes: descriptionHash>
  <32 bytes: goalAmount, e.g. 5 ETH = 0x4563918244f40000 padded>
  <32 bytes: deadline timestamp>
```

The `donate` function is `payable`, so the ETH value is passed in the transaction's `value` field rather than as a function argument. The function signature is `donate(uint256,bytes32)` – the campaign ID and an optional message hash.

All function selectors and encoding rules are documented in `lib.sh`, and the demo script uses descriptive variable names for every hex value so readers can trace what each call does.

7. Results

Summary

The demonstration completed both scenarios successfully on a local Hardhat node.

Operation	Gas Used	Description
Deploy contract	1,889,158	One-time deployment cost
Create campaign	168,133	Campaign struct storage allocation
Donate (1st donor)	185,313	First donation to campaign, new storage slot
Donate (2nd donor)	130,829	Subsequent donation, existing array append
Donate (3rd donor)	130,829	Consistent with 2nd donor
Allocate (1st)	168,449	Allocation struct storage
Allocate (2nd)	134,249	Subsequent allocation
Disburse (1st)	205,861	Includes ETH transfer to recipient
Disburse (2nd)	191,585	Subsequent disbursement
Finalize campaign	31,003	Status flag update
Refund	86,994	Includes ETH transfer back to donor
Query (off-chain via eth_call)	0	All view functions are free when called externally

Gas Costs

Campaign creation costs 168,133 gas – the initial storage allocation for the campaign struct. Donations cost 130,829–185,313 gas. The first donation to a campaign costs more for two reasons: initializing the donations array (writing the length slot for the first time), and storing a non-zero message hash (donors 2 and 3 pass a zero hash, which is cheaper to store in a freshly allocated struct slot). Allocations cost 134,249–168,449 gas, and disbursements cost 191,585–205,861 gas (higher than allocations because they include the actual ETH transfer to the recipient). Refunds cost 86,994 gas – higher than other write operations because writing the `totalRefunded` field from zero to a non-zero value incurs a cold SSTORE cost.

At a gas price of 20 Gwei and an illustrative ETH price of $3,000, this is roughly $80 for the entire campaign lifecycle including deployment, three donations, two allocations, two disbursements, and finalization.

For campaigns with many small donors, the per-donation gas cost could become significant. A campaign with 1,000 donors would incur approximately $790 in gas costs for the donation transactions alone. This is a strong argument for deploying on a Layer 2 network where gas costs are 10-100x lower (see Section 8).

Query operations – the most frequent use case for transparency – cost zero gas. Anyone can read the full audit trail at no cost.

Observations

The escrow model works as designed. Donated ETH is held in the contract and only moves when the charity explicitly disburses it. This creates a clear, auditable separation between receiving funds and spending them.

The evidence hash model is practical but relies on external infrastructure. Every allocation and disbursement includes an evidence hash, but the protocol does not address where the evidence documents are stored or how they are retrieved. A production system would need a content-addressed storage layer (IPFS, Arweave, or a dedicated evidence store) that maps hashes to documents.

Refund mechanics require time manipulation for testing. The refund scenario uses Hardhat's `evm_increaseTime` to advance the blockchain's clock past the campaign deadline. This is a standard testing technique but highlights that the refund mechanism depends on accurate block timestamps – a consideration for production deployment where block timestamps have limited precision and can be slightly manipulated by validators.

Gas costs are reasonable for high-value campaigns, marginal for high-volume campaigns. The per-operation costs are acceptable for campaigns managing significant funds (thousands of dollars or more), where the gas cost is a small percentage of the total. For campaigns with many small donations, Layer 2 deployment is the practical path.

Contract size is well within limits. The deployed bytecode is approximately 9 KB, well under Ethereum's 24,576-byte contract size limit. This leaves substantial headroom for future extensions without needing proxy patterns or contract splitting.

8. Alternative Blockchain Platforms

Ethereum mainnet provides the strongest security and decentralization guarantees but also the highest transaction costs. For a donation tracking protocol where query operations are free and write operations are infrequent (relative to queries), the cost structure is manageable for medium-to-large campaigns. For high-volume use cases with many small donations, alternative platforms offer meaningful tradeoffs.

Ethereum Layer 2 networks (Arbitrum, Optimism, Base) offer the most practical deployment path. These networks inherit Ethereum's security guarantees while reducing transaction costs by 10-100x. A donation that costs $0.09 on Ethereum mainnet would cost less than $0.01 on Arbitrum. The contract would require no modifications – Solidity contracts deploy identically on any EVM-compatible network. The tradeoff is that Layer 2 networks have slightly different trust assumptions (transaction ordering depends on the sequencer) and may have less mature tooling for some use cases. For a donation tracking protocol, these tradeoffs are acceptable.

Polygon provides another EVM-compatible option with very low transaction costs and a large ecosystem. It operates as a sidechain with its own validator set, which means different security assumptions than Ethereum mainnet or L2 rollups. For a transparency protocol where the primary concern is data availability rather than high-value DeFi operations, Polygon's security model is appropriate.

IOTA Rebased offers feeless transactions on its mainnet, which makes it particularly attractive for high-volume donation tracking where gas costs are a concern. The contract would need to be reimplemented in Move (IOTA's smart contract language). IOTA's object-based state model would change the storage pattern – campaigns could be individual shared objects rather than entries in a mapping – but the protocol logic would remain identical. The feeless model means that even campaigns with thousands of donors would incur no per-transaction costs, making the protocol accessible to small-scale community fundraising.

Solana provides high throughput and low transaction costs (fractions of a cent per transaction). The contract would need to be reimplemented using Solana's program model (Rust/Anchor). Solana's architecture is well-suited for high-frequency operations, but its account model and programming patterns differ significantly from EVM. The ecosystem is large and the tooling is mature, but the developer migration cost is higher than for EVM-compatible chains.

Cosmos / CosmWasm enables purpose-built blockchains with custom governance and fee structures. An application-specific blockchain for donation tracking could eliminate transaction fees entirely and add domain-specific features (e.g., built-in charity verification). The development cost is significantly higher, but the result is a purpose-built platform rather than a general-purpose smart contract.

The most pragmatic recommendation for a production deployment is an Ethereum Layer 2 network. The contract works without modification, the tooling is identical, the costs are dramatically lower, and the security guarantees are strong. IOTA is the most attractive option for organizations that need feeless transactions and are willing to invest in a Move implementation.

9. Future Directions

Stablecoin Support

The most practical near-term extension is accepting stablecoins (USDC, USDT, DAI) instead of or alongside ETH. Donors overwhelmingly think in fiat terms, and ETH's price volatility makes it poorly suited for charitable giving at scale. Stablecoin donations would provide a stable unit of account while retaining the smart contract escrow and transparency benefits. The contract modification is straightforward: replace `msg.value` with ERC-20 `transferFrom`, and add an approved token whitelist.

Milestone-Based Release

The current protocol allows the charity to allocate and disburse freely, recording every decision transparently. A more restrictive model would gate fund release on verified milestones – the charity commits to specific outcomes, and funds are released only when those outcomes are confirmed. Confirmation could come from human validators (similar to Alice.si's model), oracle networks, or AI-assisted evidence analysis. This model increases accountability but also increases complexity and the risk of funds being locked indefinitely if validators are unavailable.

Donor Governance

A lightweight governance extension would allow donors to vote on allocation priorities (but not operational decisions). Donors could express preferences for how campaign funds are distributed across purposes, and the charity would consider these preferences alongside its operational expertise. This does not give donors veto power but creates a feedback channel that incentivizes alignment between donor intent and charity action.

Multi-Signature Disbursement

For high-value campaigns, disbursement could require multiple signatures – for example, the charity executive director, a board member, and an independent auditor must all approve before funds move. This adds a layer of internal control that the current single-creator model lacks.

Regulatory Integration

Charities operate within regulatory frameworks that require specific reporting. An extension could generate regulatory-compliant reports directly from on-chain data – donor contribution summaries for tax receipts, program expenditure breakdowns for Form 990 filing, and grant utilization reports for institutional funders. The on-chain data is structured enough to support automated report generation.

Fiat On-Ramp Integration

The ultimate adoption barrier is the requirement for donors to hold ETH or cryptocurrency. Fiat on-ramp integration (through payment processors that convert fiat to crypto and execute the on-chain donation in a single transaction) would make the protocol accessible to donors who have never used a blockchain. Services like MoonPay, Transak, and Wyre provide this infrastructure.

Production Hardening

The proof-of-concept contract is designed for clarity and auditability rather than defense-in-depth. A production deployment would benefit from several additional safeguards. A `nonReentrant` modifier (from OpenZeppelin's `ReentrancyGuard`) on `disburse` and `refund` would close off reentrancy as a class of vulnerability, even though the current CEI ordering does not expose an exploitable path. A professional security audit is essential before handling real funds – the contract has not been audited. Gas optimization through batched operations (e.g., a `batchDonate` function for campaigns with many small contributors) would reduce per-transaction overhead on mainnet.

Appendix A: Project File Reference

Repository Structure

```
transparent-charitable-donations/
|-- README.md
|-- .env.example                    # Environment configuration template
|-- run_demo.sh                     # End-to-end demo script
|-- contracts/
|   `-- DonationTracker.sol         # Solidity contract (0.8.28)
`-- demo/
    |-- lib.sh                      # Shared helper functions (RPC calls, ABI encoding, formattir
    `-- sample_data.json            # Campaign descriptions and evidence text for demo scenarios
```

Environment Variables

Variable	Required	Default	Description
`RPC_URL`	Yes	`http://localhost:41545`	Ethereum JSON-RPC endpoint
`ETH_STARTER_PACK_DIR`	Yes	`../Ethereum-Local-Testing-Starter-Pack`	Path to the starter pack (for compilation and account creation)

Account addresses are not configured manually. The demo script creates six fresh accounts at startup using the starter pack's `./eth.sh create-account` command and impersonates them on the Hardhat node for transaction signing.

Appendix B: Full Demonstration Output

B.1: Clean Water Initiative Scenario

```
============================================================
  Setup: Creating Accounts
============================================================

-> Creating funded accounts via starter pack...
   Charity: 0x05C344828170Ea4b9635e5d6a2c54865c2Cc7fb5
   Donor 1: 0x14510963A36488987825f4e91A48a5d3a3004219
   Donor 2: 0x484253b55e2646038AE7ff0e89e68A970684330C
   Donor 3: 0xA035aE57fd92742404Bc728F276CAB81A5E60711
   Recipient 1: 0xd61E1e42A0d03b2DFEef35558363bEF42924B999
   Recipient 2: 0x85aD0d4C9Df0A5F0E3d7711e0e64C8a02e935061

============================================================
  Setup: Compiling and Deploying Contract
============================================================

-> Compiling DonationTracker.sol via starter pack hardhat container...
-> Deploying DonationTracker...
   Deployed. Gas used: 1889158
   Contract: 0xce6aaf524b62c14f320529d08b04e4659d11cc85

============================================================
  Scenario 1: Clean Water Initiative
============================================================

-> Step 1: Creating campaign...
   Description: "Clean Water Initiative - Providing clean drinking
                 water to underserved communities in East Africa
                 through portable filtration units distributed
                 to rural villages."
   Description hash: 0x1b07ce475e309dd52055c2dfb6b00b156d7dd8e1a06d30bd9067131ae1ccd383
   Goal: 5 ETH
   Deadline: 1779192182 (30 days from now)

   Campaign ID: 0
   Created. Gas used: 168133

-> Step 2: Donor 1 contributes 2.0 ETH...
   Donor: 0x14510963A36488987825f4e91A48a5d3a3004219
   Amount: 2.0 ETH
   Message: "For the children"
   Message hash: 0x742d0a925d0b27f67552e76266603346af4525bb4bbfee7451c38bb60ec61309

   Donated. Gas used: 185313

-> Step 3: Donor 2 contributes 1.5 ETH...
   Donor: 0x484253b55e2646038AE7ff0e89e68A970684330C
   Amount: 1.5 ETH
```

```
   Donated. Gas used: 130829

-> Step 4: Donor 3 contributes 2.0 ETH...
   Donor: 0xA035aE57fd92742404Bc728F276CAB81A5E60711
   Amount: 2.0 ETH

   Donated. Gas used: 130829

-> Step 5: Query campaign status...
   Campaign ID: 0
   Creator: 0x05C344828170Ea4b9635e5d6a2c54865c2Cc7fb5
   Goal: 5.0000 ETH
   Total donated: 5.5000 ETH
   Total allocated: 0.0000 ETH
   Total disbursed: 0.0000 ETH
   Status: Active
   Balance held: 5.5000 ETH

-> Step 6: Allocate 3.5 ETH to Water purification equipment...
   Purpose: "Water purification equipment - 50 portable filtration
             units for distribution to villages in the Rift Valley
             region. Supplier: WaterTech Supplies Ltd."
   Purpose hash: 0xc2dcdd8b5d40c58c73ae9b9f7348a551c023553be64033c3c36ddc2587f93765
   Evidence: Purchase order PO-2026-0417 from WaterTech Supplies Ltd.
   Evidence hash: 0x20aa2ddfb28ae86f031a01caff30209ed58636589ffebb5e4630bafea998ca0f

   Allocated. Gas used: 168449

-> Step 7: Allocate 1.5 ETH to Distribution logistics...
   Purpose: "Distribution logistics - Transport and installation
             of filtration units across 12 villages in the Rift
             Valley region. Contractor: Regional Transport Co."
   Purpose hash: 0xb931cbf88337903e11edbafaef3afbc9806a10e52a73f6dd2fc907596fab6817
   Evidence: Logistics contract LC-2026-0089 with Regional Transport Co.
   Evidence hash: 0xaee3caf5a312a65a84d01406f03a85e2c82af1c0efcf9f5c36d05c36b4805152

   Allocated. Gas used: 134249

-> Step 8: Query allocations...
   Allocations: retrieved (320 bytes of ABI data)

-> Step 9: Disburse 3.5 ETH to WaterTech Supplies...
   Recipient: 0xd61E1e42A0d03b2DFEef35558363bEF42924B999
   Amount: 3.5 ETH
   Allocation: 0 (Water purification equipment)
   Evidence: Delivery receipt DR-2026-0417
   Evidence hash: 0xe71044cb5eb2d8ae0086c63f9eaa95aebf8498dafab8bd5d3f913922136fa9d5

   Disbursed. Gas used: 205861

-> Step 10: Disburse 1.5 ETH to Regional Transport Co....
   Recipient: 0x85aD0d4C9Df0A5F0E3d7711e0e64C8a02e935061
   Amount: 1.5 ETH
   Allocation: 1 (Distribution logistics)
   Evidence: Transport completion certificate TC-2026-0089
   Evidence hash: 0x0a0c46f683f3f5cfb7dae7b4e2cd24c4b85ad4ab9d3e161dba8052f1eff1944f
```

```
   Disbursed. Gas used: 191585

-> Step 11: Finalize campaign...
   Finalized. Gas used: 31003

-> Step 12: Full audit trail query...
   Campaign ID: 0
   Creator: 0x05C344828170Ea4b9635e5d6a2c54865c2Cc7fb5
   Goal: 5 ETH
   Total donated: 5.5000 ETH
   Total allocated: 5.0000 ETH
   Total disbursed: 5.0000 ETH
   Remaining balance: 0.5000 ETH (unallocated surplus)
   Status: Finalized

   Flow summary:
     Donated:    5.5000 ETH (3 donors)
     Allocated:  5.0000 ETH (2 purposes)
     Disbursed:  5.0000 ETH (2 recipients)
     Remaining:  0.5000 ETH (held in contract)

============================================================
  Scenario 1: COMPLETE
============================================================
```

B.2: Failed Campaign with Refund

```
============================================================
  Scenario 2: Failed Campaign (Refund)
============================================================

-> Step 1: Creating campaign with high goal...
   Description: "Emergency Shelter Program - Temporary housing for
                 families displaced by flooding in coastal regions."
   Goal: 100 ETH
   Deadline: 1776600252 (short deadline for demo)
   Campaign ID: 1

   Created. Gas used: 151045

-> Step 2: Donor 1 contributes 1.0 ETH...
   Donor: 0x14510963A36488987825f4e91A48a5d3a3004219
   Amount: 1.0 ETH

   Donated. Gas used: 165041

-> Step 3: Advance time past deadline...
   Using evm_increaseTime to advance 120 seconds...
   Mining a block with evm_mine...
   Time advanced. Block timestamp is now 1776600314
   (past deadline 1776600252).

-> Step 4: Attempt refund (goal not met, deadline passed)...
```

```
   Donor: 0x14510963A36488987825f4e91A48a5d3a3004219
   Refund amount: 1.0 ETH

   Refunded. Gas used: 86994

-> Step 5: Verify campaign balance...
   Campaign balance: 0.0000 ETH
   Donor successfully refunded.

============================================================
  Scenario 2: COMPLETE (Refund successful)
============================================================

============================================================
  Summary
============================================================

Scenario 1 (Clean Water Initiative):
  Campaign created, funded by 3 donors (5.5 ETH total)
  Allocated to 2 purposes (5.0 ETH)
  Disbursed to 2 recipients (5.0 ETH)
  Full audit trail queryable at zero gas cost

Scenario 2 (Failed Campaign):
  Campaign created with unmet goal
  Time advanced past deadline
  Donor successfully refunded
```

This paper accompanies the Transparent Charitable Donation Tracking proof of concept, developed by Consensix Labs. The code is provided for research and educational purposes. It has not been audited for production use.