A Decentralized Crowdfunding Platform for Transparent and Anonymous Donations

PRATHAMESH KOLHE[1], BHUSHAN DHAWAN[2], SNEHA GHOSH[3], NEHA RATHOD[4]
[1, 2, 3, 4]*MIT Art, Design and Technology University*

Abstract- ***The design and deployment of a decentralised crowdfunding web application that permits transparent but private bitcoin donations is presented in this paper. Our platform consists of a React/Tailwind frontend, Thirdweb SDK for Web3 integration, and an Ethereum smart contract written in Solidity to manage campaigns and contributions. ETH is donated to campaigns by donors using their wallets (like MetaMask) without disclosing any personal information other than their pseudonymous address. While the lack of centralised accounts maintains donor anonymity, the blockchain's immutable ledger documents every transaction, guaranteeing confidence and transparency of financial flows 1 2. The implementation of key features (campaign creation, donation, and data retrieval), deployment on Ethereum testnets, and system architecture (smart contract, backend, and frontend stack) are all covered in detail. We show testing results and examine privacy/security issues (such as pseudonymity vs. traceability 3). The outcomes show that our platform preserves donor anonymity while achieving on-chain accountability for money. We detail next work to improve privacy and performance and talk about limitations (such as scalability and anonymity constraints 4).***

I. INTRODUCTION

Many people may now fund projects online thanks to crowdfunding, which has democratised access to capital. However, traditional centralised platforms frequently struggle with transparency and trust. Central middlemen have the potential to mismanage funds, postpone projects, or conceal project progress. Blockchain-based crowdfunding, on the other hand, uses smart contracts and an immutable record to impose regulations and offer transparent accountability 1 2. According to earlier research, for instance, "blockchain's immutable ledger ensures transparency and trust, as all transactions and project details are tamper-proof" 1; smart contracts, on the other hand, can automate fund distributions based on predetermined criteria, lowering fraud and boosting contributor trust 2–5. Our objective is to create a decentralised platform that respects donor privacy and embraces these benefits. Contributions of cryptocurrency are made anonymously (without requiring personal registration), and the chain makes all transaction information public. We use a contemporary frontend (React with Tailwind CSS) that communicates over Web3 frameworks and the Thirdweb SDK, as well as a Solidity smart contract for campaign management. With a focus on the platform's technical aspects and its more general objectives of transparency, decentralisation, and user privacy, this paper outlines the system architecture, implementation specifics, and evaluation of the platform. Related Work

Recent studies have given decentralised crowdfunding a lot of attention. Numerous studies emphasise that trust, security, and openness are essential components of a successful blockchain crowdfunding campaign. 6. For example, Loso et al. discover that platform success is significantly enhanced by security and transparency. 6. Similarly, More et al. stress that blockchain systems can lower fraud and facilitate trustless transactions between project founders and contributors 2 by utilising smart contracts and a tamper-proof ledger. 5. Direct peer-to-peer fundraising is made possible by decentralised systems, which do away with middlemen and make all contributions publicly visible. Donors can monitor how their money is being used because to this visibility, which promotes accountability.

"Excessive transparency may expose contributor identities and transaction details, raising privacy concerns," according to studies, which also highlight a

trade-off between transparency and privacy. 3. Although donors in blockchain systems are anonymous (represented solely by addresses), it may be possible to connect addresses to actual people through advanced analysis. New platforms try to solve this; for instance, ETHGlobal's Crowdera project allows anonymous donations without wallet monitoring. 9. By enabling anonymous contributions (addresses only) and providing donors with on-chain guarantees about money utilisation, our work continues this trend. Additionally, we are in line with previous work that employs Ethereum/Hardhat for smart contracts 11 and React.js and Node.js for the web interface and server-side logic 10.. In conclusion, current research highlights blockchain's potential for safe, transparent crowdfunding 1 2; our solution expands on these findings by fusing established smart contract designs with cutting-edge Web3 tools to strike a compromise between privacy and transparency. System Design and Architecture

- A frontend web application and a blockchain backend (Ethereum smart contract) make up the platform's standard Web3 dApp architecture. All campaign data is contained in the smart contract (in Solidity), which also manages fundraising logic. Using the deployment tools from Thirdweb and Hardhat, we set it up on the Ethereum Goerli testnet. Tailwind CSS was used to style the frontend, which is a single-page application created with React. Contract calls, wallet connectivity, and blockchain interaction are made easier with Thirdweb's React SDK and ethers.js frameworks. The architecture is hypothetically depicted in Figure 1 (not shown):
- Blockchain Layer: The essential component of the system is the Crowd Funding contract. All campaign data is stored in a Campaign struct, which is mapped by campaign ID and has the following fields: owner address, title, description, target amount, deadline timestamp, image URL, donors list, and donations list. The main duties consist of: To start a new campaign in storage, enter create Campaign (owner, title, description, target, deadline, picture) and confirm that the deadline is in the future.
- donate To Campaign(id) payable: sends the Ether to the campaign owner after storing the donor's address and donation amount for the campaign id.
- arrays of donor addresses and gift amounts are returned by the getDonators(id) view. All campaigns are returned for presentation via the getCampaigns() view function. Decentralised control is enforced by these smart contract features: donation records are permanently recorded on-chain 5 and payments are automatically transferred to owners.
- Contract Deployment: Using Infura RPC and a private key, we set up Hardhat (solidity 0.8.9) to build and deploy to Goerli. Contract identification and release were accomplished using Thirdweb CLI. Gas expenses were decreased by turning on optimiser settings. Contract updates were made easier by this automated toolchain.
- Frontend Stack: @thirdweb-dev/react hooks are used by the React frontend. A central StateContext provider connects to the deployed contract address using useContract('0x...'), enables writes using useContractWrite, and retrieves the connected wallet using useAddress. Page navigation is offered by React Router (Home, CreateCampaign, CampaignDetails, Profile pages). Tailwind CSS tools are used to create user interface elements such as loaders, buttons, and custom cards. With the use of Thirdweb's hosting tools, the frontend can be installed on any web server or IPFS as a static build that Vite bundles.Web3 Integration: MetaMask (or any injected Ethereum wallet) is used for user authentication. Upon connection, the user’s address is recognised by the app. Contract calls are transmitted as transactions, such as createCampaign and donateToCampaign. Reads are used to activate the data retrieval functions (getCampaigns, getDonators). Although Thirdweb abstracts away part of the complexity, ethers.js is responsible for sending and signing transactions.

All things considered, our approach guarantees transparency (all transactions are on-chain) and decentralisation (no server maintains funds). Our architecture adheres to the popular usage of React and Node.js in such dApps 10.

III. IMPLEMENTATION DETAILS

The Solidity smart contract (Crowd Funding. sol), the Node.js/Hardhat configuration, and the React frontend

make up the three primary components of the implementation.

The Crowd Funding. sol smart contract: To hold them, we created a mapping campaign and a campaign struct. IDs are tracked by a global counter number Of Campaigns. We record the updated campaign details in create Campaign after requiring that the specified deadline is in the future (intended check). The function in donate To Campaign is payable; it enters message. sender and message. value into the arrays of donors and donations for the campaign. With a low-level call, it then instantly sends the Ether to the campaign owner, updating amount Collected if it is successful. Campaign or donor lists are returned by utility view functions. For auditing purposes, all state-modifying operations implicitly produce logs.

Local Development (Hardhat/Thirdweb): Hardhat was utilised for compilation and testing locally. In order to save gas, the hardhat.config.js file includes optimiser settings (runs=200) and designates Goerli as the default network. We inserted @thirdweb-dev/ contracts (even if our contract is custom) and scripts in package. json to identify and implement the contract using Thirdweb's SDK. The deploy script, for instance, launches thirdweb deploy, which provides an address and uploads the contract to Goerli.

Frontend State Management: A State Context Provider covers the application in React, offering state and global functionalities. The contract is loaded using use Contract (ADDRESS). We destructure mutate A sync: create Campaign using use Contract Write, binding to the create Campaign method of the contract. After formatting values (such as converting deadlines to timestamps or Ether to Wei), the publish Campaign method calls create Campaign to send the transaction. Likewise, contract. Call ('donate To Campaign', [id], { value: ethers. utils. parse Ether(amount) }) is called by a contribute function.

Frontend Components and Pages: The Home page uses the context's get Campaigns() to load all campaigns on mount. This function parses Big Number values to strings and calls the contract's get Campaigns. A list of Fund Card components is rendered by the Display Campaigns component, which is used to display campaigns. Each card leads to Campaign Details when clicked, and it displays the campaign title, owner, and amount raised. A "Submit" button that invokes create Campaign is included on the CreateCampaign page, along with a form with the following fields: Title, Description, Target ETH, Deadline, and Image URL. A list of donors along with their amounts (retrieved via get Donations) and campaign details (picture, progress bar, number of days left, and total backers) are displayed on the Campaign Details page. Additionally, it features a "Fund Campaign" button that initiates donations and a donation entry field. Campaigns are filtered by owner == user Address on the Profile page. displaying only those that were made by the current user with the utility get User Campaigns. Tailwind CSS classes are used for responsive style in UI elements.

Utilities: We incorporate assistance functions, such as calculate Bar Percentage for progress and days Left to determine the number of days left from the timestamp. Ethers. utils. format Ether is used to format ether values. Additionally, the algorithm verifies the validity of an image URL before to submission.

This implementation makes use of common tools, such as Thirdweb SDK to streamline contract access, ethers.js for Ethereum types and utilities, and React and Node.js for the app structure 10. All important data (donations, campaign definitions) must be kept on-chain, thanks to Solidity and Web3 code. The code structure closely resembles the project that was uploaded: UI elements represent contract data fields, and smart contract methods correspond to frontend calls.

IV. PRIVACY AND SECURITY CONSIDERATIONS

Our platform strikes a balance between user privacy and transparency. Every gift and campaign creation is documented on the public Ethereum ledger, ensuring transparency. By contacting the smart contract (using our frontend or other methods), anyone can view the contributor lists or campaign mapping. This type of transparency "allows contributors to track how their funds were being used, ensuring that funds were not misused," according to earlier study (8 3). In fact, our system allows donors to view the precise amounts

raised as well as the campaign owner's name, which fosters trust.

The pseudonymous nature of blockchain addresses, however, offers privacy. Donors just link their cryptocurrency wallet and sign a transaction; they never provide names or email addresses. As a result, on chain data only displays an amount and an Ethereum address (such as 0xABC123). This provides a degree of anonymity that is comparable to what proponents of crypto privacy anticipate. 3. We don't employ KYC or maintain any off-chain personal information. We do note, though, that blockchain technology is pseudonymous and not completely anonymous; if a user has publicly marked an address elsewhere, on-chain analysis may associate it with an identity. 3. It is acknowledged to be difficult to maintain anonymity "without compromising system integrity." Although complete anonymity would necessitate more sophisticated methods (such as mixers or ZK-proofs), which are outside the purview of the current system, we reduce risk by not requiring any identification at all.

There are various approaches to security. To minimise errors, the smart contract has a simple, low-complexity architecture. When transferring ETH via call, we take care to ensure success by verifying the returned bool before updating the state. Overflow checks are incorporated into Solidity 0.8.x, which is what we use. More validation should preferably be included in the code, though (for example, make sure the input parameter is used by create Campaign's deadline check, as said). We reduce the dangers of smart contract vulnerabilities (such as reentrancy or integer overflow) by prohibiting arbitrary fallback logic or loops over untrusted data in our simple contract. Security is carried over from MetaMask and Ethereum to the frontend and wallet side: user signatures are required for transactions, while Web3 libraries such as Thirdweb and Ethers use well-known crypto for key management.

Trade-offs should be noted: we rely on the blockchain's security model, which is generally reliable, but loss could result from user error (such as transferring to the incorrect address) or smart contract problems. User education (verifying information in MetaMask prompts) and routine audits can be beneficial. Furthermore, unlike centralised systems, we have no way to identify malevolent actors or freeze funds due to the privacy measure of anonymity. These issues will be covered in more detail as restrictions.

V. DEPLOYMENT AND TESTING

For testing, we put the smart contract on the Ethereum Goerli testnet. After doing preliminary testing on a local node using Hardhat, we set up the Goerli network using an RPC URL (Ankr) and a private key from environment variables. The deployment, which automated contract publishing, was carried out using npx thirdweb deploy. The frontend configuration included the final contract address. The React application was constructed and could be hosted on IPFS using Thirdweb's storage or served locally (although we chose localhost for testing).

Several test accounts on Goerli were used for functional testing. We confirmed the end-to-end processes of establishing a connection with MetaMask, generating a new campaign with legitimate inputs, and generating donations. Every transaction (contribution or campaign creation) was mined on Goerli and required a signed MetaMask confirmation. On the testnet, we saw standard Ethereum block timings (15s) and petrol prices of a few Gwei. The system performed flawlessly when handling dozens of transactions. It was almost instantaneous to fetch data (such as by executing getCampaigns).

The usability of the user interface was evaluated; during transactions, the UI displays the proper loading states and asks users to link their wallet (using use Metamask). The user interface (UI) is updated upon successful contract calls (e.g. freshly launched campaigns display on the Home page). We observe that, despite being a static React application, the UI was responsive and styled with Tailwind CSS, offering a contemporary appearance with little load overhead.

As a practical aside, we set up the contract calls in the frontend to target Goerli. The Ethereum mainnet address (or another chain) would be used for a mainnet launch. Additionally, Thirdweb's upload function might publish the build to a CDN with a site URL in

order to deploy the frontend itself. Considering our current configuration, these tasks are simple.

VI. RESULTS AND DISCUSSION

Our platform effectively accomplishes its main objectives. On-chain, all campaign information and donation transactions are openly accessible. Because of this transparency, anyone—including donors—can check the overall amount of money raised for a campaign and see the amount and originating address of each gift. For instance, the blockchain entry indicates that donations[3][i] = 0.1 ETH and donators[3][i] = 0xUserAddress following a user's donation of 0.1 ETH to Campaign #3. As mentioned in related work 8 1, such a record cannot be altered once it has been mined, which fosters confidence.

However, donor identities are kept secret outside of their wallet address. We note that our design does not gather any personal information, which is consistent with the idea that "blockchains create security and trust, as they eliminate the need for an intermediary" 7. However, we also recognise that donors may lose their anonymity if their address is connected to another organisation. For users who value pseudonymity, this degree of privacy is appropriate and comparable to typical cryptocurrency donations. The system doesn't leak any emails or names. In terms of scalability, the prototype is modest. On Ethereum's current main net (or even testnet), throughput is limited, so the platform is best suited for projects with moderate traffic. The contract functions like get Campaigns() iterate over all campaigns, It would be expensive for hundreds or more entries, but acceptable for tens. If necessary, off-chain indexing (for example, using The Graph) could lessen this. We also observe that real deployment would be impacted by gas prices (e.g., gas for a create Campaign or give transaction); economics could be improved by contract optimisation or layer-2 chains.

All things considered, our findings show that the platform protects user privacy at the address level while offering a verifiable record of crowdfunding activity. Contributors may "track how their funds were being used" on an open ledger 8, which increases accountability. These results are in line with earlier research. We accomplish the hailed advantages of decentralised crowdfunding by doing away with centralised control and depending on smart contracts.

VII. LIMITATIONS

Despite its functionality, the current system has several limitations:

- Anonymity Limits: As previously said, blockchain is pseudonymous. Anonymity is lost if a donor's wallet is ever linked to their actual identity (for example, through an exchange KYC or a publicly displayed address). Advanced privacy techniques, such as mixers and zero-knowledge proofs, are not used by our software to protect transactions. Therefore, privacy is restricted to anonymity at the standard address level.
- Security and Code Risks: The smart contract is straightforward, but any errors could have serious consequences. For example, the need function in the createCampaign function was designed to verify the input deadline; but, in its current form, it compares an uninitialised field, so evading validation. Such problems need to be examined and fixed. Generally speaking, smart contract mistakes might lead to vulnerabilities, hence extra caution (or official audits) is required 4.
- Scalability: Under high load 12, Ethereum's throughput (about 10–30 TPS) limits the system. For large campaigns, the contract's linear data structures (donor arrays) may also encounter gas or return size limitations. Without protocol-level adjustments, these issues could prevent the platform from growing to millions of donors.
- Regulatory and Usability IssuesWhile using cryptocurrencies to donate circumvents some restrictions, it may also expose one to others (such as those pertaining to donor protections and tax compliance). Additionally, requiring a cryptocurrency wallet can turn off non-technical consumers. These are more general restrictions on blockchain crowdfunding.
- Functionality: Donations are currently sent to the owner directly under the contract, with no escrow or refund requirements. There is no implementation of features like campaign cancellation, milestone-based release, or reward systems.

These limitations suggest areas for improvement, as discussed next.

VIII. FUTURE WORK

Future enhancements can address the above limitations and extend functionality:

- Enhanced Privacy: One may incorporate privacy layers to genuinely anonymise donations. Transaction links would be obscured, for instance, by implementing zero-knowledge proofs (zk-SNARKs) or coin-mixing algorithms. Similar concepts could reward contributors without disclosing their identities. Some projects also mint anonymous NFT badges for donations 9. This extension may be informed by research on safe anonymous contribution mechanisms 13.
- Advanced Smart Contract Features: Make the payment logic more complex. For instance, provide a time lock or multisignature withdrawal feature, set campaign milestones that only release cash upon reaching certain targets, or let donors vote on how the funds are used. Support for ERC-20 tokens (allowing donations of stablecoins) could reduce cryptocurrency volatility.
- Scalability Solutions: Use layer-2 networks (like Polygon or Optimism) to boost throughput and lower costs. To manage a large number of campaigns and donors effectively, use an accompanying database or off-chain indexing services (The Graph).
- User Experience: Enhance the user interface and user experience for non-crypto users, perhaps by adding explanatory manuals or fiat on-ramps. Include functions like social sharing, campaign updates, and notifications. Accessibility would be expanded with multilingual support and smartphone optimisation.
- Security Audits and Testing: To guarantee robustness, conduct official audits of the smart contract code. 4. To detect regressions, incorporate continuous integration and unit testing. Put in place automated contract monitoring (e.g., warnings on significant transactions).
- Community Feedback and Governance: Think about governance tools (like a DAO) for fee structures or platform regulations. To improve platform needs and trust presumptions, involve early users.

Exploring these directions will further align the platform with the principles of secure, private, and scalable decentralized crowdfunding.

CONCLUSION

We have introduced a completely decentralised crowdfunding tool that uses contemporary web technologies and Solidity smart contracts to enable anonymous and transparent cryptocurrency donations. All campaigns and donations are clearly recorded via the blockchain smart contract, which also provides an auditable ledger 8. React and Tailwind were used to build the frontend, which communicates with Thirdweb's Web3 SDK to let users create campaigns and make donations using their cryptocurrency wallets. Key goals are achieved by this design: donors can stay anonymous while knowing that their money will go to the right person. The feasibility of blockchain for trustworthy fundraising is demonstrated by our implementation, which supports earlier findings that safe, open decentralised crowdfunding may empower both project creators and contributors 1 2. However, we are aware of issues like residual privacy leakage and on-chain scalability. 4. 3. Future research will concentrate on enhancing functionality (governance, multi-currency) and scalability and privacy (e.g. ZK-proofs, Layer-2 deployment). To sum up, our initiative contributes to the larger trend of decentralised, trust-minimized crowdfunding by demonstrating a workable compromise between openness (via immutable records) and user privacy (through anonymous crypto transactions).

Acknowledgment: We thank the Thirdweb and Ethereum communities for tooling and documentation, and the anonymous donors and testers who provided feedback during development.