Smart Contract Coordinated Privacy Preserving Crowd-Sensing Campaigns

Delia Cavalca Chiara Anna Cartarasa Giuseppe Cacciapuoti

delia.cavalca@studio.unibo.it chiaraanna.cartarasa@studio.unibo.it

giuseppe.cacciapuoti@studio.unibo.it

Alma Mater Studiorum - University of Bologna

Academic Year 2024-2025

1 Introduction

Crowdsensing is a data collection method that relies on the active participation of users, who contribute information through connected devices such as smartphones and IoT sensors. This approach offers several benefits, including large-scale data collection and lower costs compared to traditional methods. However, it also presents challenges, particularly in terms of **security, reliability, and privacy**. Centralized servers are vulnerable to attacks, data quality can be compromised by intermediaries, and user information must be adequately protected.

To address these issues, we have developed a **decentralized crowdsensing DApp** that ensures secure, transparent, and tamper-proof data sharing.

Our solution is designed to eliminate intermediaries by leveraging decentralized technologies that improve both security and reliability. Key features of the system include:

- **Decentralized Storage with IPFS:** Data is distributed across multiple nodes and stored using the InterPlanetary File System (IPFS), ensuring that it is not dependent on a single central server. This enhances data availability and resilience against failures or attacks on centralized infrastructures.
- **Traceability and Transparency with Blockchain and Smart Contracts:** All transactions are recorded on a blockchain, ensuring a tamper-proof, transparent, and verifiable system. Smart contracts manage transactions without the need for intermediaries, ensuring consistency and automatic execution of predefined conditions.

- **Data Security through Encryption, Anonymization, and Digital Signatures:** All data is encrypted before being uploaded to IPFS, ensuring that only those with the decryption key can access it. Data is divided into uniformly sized blocks, providing additional protection and facilitating efficient storage. Digital signatures are used to verify the authenticity and integrity of the data, ensuring that any tampering is immediately detectable.

This combination of technologies creates a robust system that addresses traditional crowdsensing challenges, providing a secure, transparent, and reliable data-sharing environment.

2 Crowdsensing Platform

Our decentralized crowdsensing platform is designed to collect, validate, and reward users for contributing data in a secure and transparent manner. The platform operates through a structured process that ensures data integrity and incentivizes participation.

The crowdsensing campaign is managed by an **Admin** user, who defines the parameters and objectives of data collection. **Users** can upload their data to the system by paying a small fee, which helps maintain the platform's sustainability. Once the data is submitted, **Verifiers** are responsible for assessing its authenticity, integrity, and consistency. They have the authority to either validate or discard submissions based on predefined criteria.

To encourage participation and ensure fairness, the platform implements a **reward and refund system**. Verifiers receive rewards for successfully completing validation tasks, while Users are reimbursed when their data is validated. This decentralized approach enhances trust, transparency, and data reliability, eliminating the need for intermediaries and ensuring a fair and efficient crowdsensing ecosystem.

3 System Architecture and Protocol

In this section, we present our system model. The data collected in our system consists of JSON files, which users can upload to the platform.

3.1 Actors

Our system is composed of the following actors:

- **Users:** Individuals who act as data sources, voluntarily providing data.
- **Verifiers:** Entities responsible for assessing the authenticity, integrity, and consistency of the submitted data, determining whether it should be accepted or rejected.

- **Admin:** Account with which the contract is deployed. He can transfer a balance to the contract, manage campaign settings, and view data collection statistics.
- **Smart Contract (SC):** A contract deployed on the blockchain that manages the crowdsensing campaign, coordinating data collection and ensuring fair distribution of rewards.
- **Decentralized File Storage (IPFS):** A secure, distributed storage system where encrypted data is stored, ensuring accessibility and protection from tampering.

3.2 Protocol

The protocol is shown in Figure 1 and is as described in the following. To simplify the explanation, we consider a single user in the description. However, in practice, multiple users act as data sources, and multiple verifiers can be involved in the validation process.

1. **Smart Contract Deployment:** The Smart Contract (SC) is deployed, officially launching the crowdsensing campaign. This step is carried out by the campaign organizer, who seeks to collect data from multiple sources.
2. **Data Collection:** A User gathers the data (.json file) that will be uploaded to the Decentralized File Storage (IPFS). The User selects the data he wants to upload.
3. **Data Encryption:** The User encrypts the data using a hard-coded encryption key. AES-256 encryption is used to encode the data.
4. **Anonymization:** The User ensures anonymity by dividing the data into uniformly sized blocks before uploading to IPFS. The data is divided into blocks of the same size.
5. **Authenticity:** The User guarantees authenticity by signing the digest. The digest is signed using the User's Private Key.
6. **Data Upload to IPFS:** The encrypted file and signed digest are uploaded to IPFS, generating the Content Identifier (CID) of the data.
7. **Data Location Communication:** Once the data is successfully stored on IPFS, the User communicates the corresponding CID (which acts as the data's unique location identifier) to the Smart Contract. The User notifies the SC and pays a submission fee.
8. **Verification Request:** The SC emits a verification request event, notifying available Verifiers that new data needs to be checked.
9. **Data Location Request:** The Verifiers ask the SC for the data locations.

10. **Data Retrieval:** The Verifier retrieves the encrypted file and its corresponding digital signature from IPFS. The Verifier also obtains the User's Public Key from the Smart Contract.

11. **Integrity and Authenticity Verification:** The Verifier calculates the digest of the encrypted file using the same SHA-256 hash function. The Verifier derives the Public Key of the signer from the calculated digest using the signature. The Verifier compares the derived Public Key with the Public Key retrieved from the Smart Contract to confirm the integrity and authenticity of the data.

12. **File Decryption:** If the verification is successful, the Verifier decrypts the file using the AES key to access the original data.

13. **Data Validations:** The Verifier analyzes the data and reports to the SC whether it is valid.

14. **Verifier Reward:** If the verification is deemed valid and trustworthy, the SC rewards the Verifier.

15. **User Refound:** If the data passes the verification, the User receives a refund. If the data fails verification, the User does not receive a refund and loses the submission fee.

16. **Campaign Completion:** Once a minimum threshold of successfully verified data is reached, the SC notifies the relevant parties, providing the locations of the verified data chunks, thereby concluding the campaign.

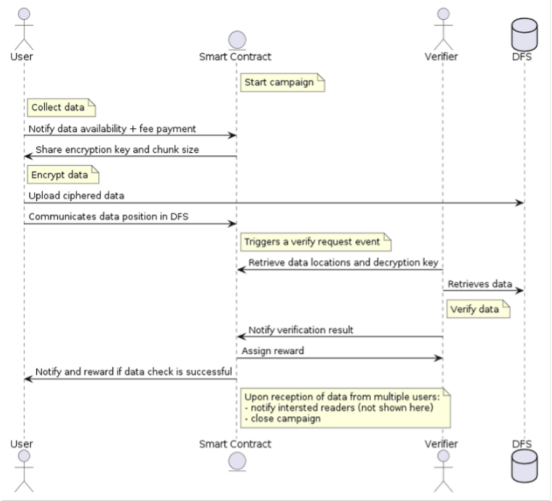

Figure 1: System Protocol

A critical security measure implemented in the system is that the encryption key is never stored on the Smart Contract or transmitted through blockchain transactions. This design decision is essential to prevent potential security vulnerabilities, such as key interception or unauthorized access. By keeping the encryption key entirely off-chain, the system ensures that only the data owner or authorized entities possess the necessary credentials to decrypt the data. This approach significantly enhances data confidentiality and minimizes the risk of exposure or tampering in a decentralized environment.

4 Technologies

This section describes the technologies used for the implementation of the described prototype.

- **Frontend:** The Vue.js framework is used to build a responsive and interactive user interface, providing an intuitive experience for users to interact with the platform. This enables seamless data uploads and interactions with the crowdsensing campaign.
- **Backend:** The backend is powered by Node.js and Express.js, allowing the creation of a robust and scalable server-side application. These technologies handle requests, manage user interactions, and efficiently integrate with blockchain and IPFS services.
- **Blockchain:** The platform is built on the Ethereum network, leveraging blockchain technology to ensure data immutability, transparency, and security. This decentralized approach eliminates centralized authorities and ensures the authenticity of the crowdsensing data.
- **Smart Contract:** The Smart Contract is written in Solidity and automates critical functions such as data validation, reward distribution, and user interaction management with the platform. The Smart Contract ensures that the rules are automatically enforced, providing a transparent and secure environment for all participants.
- **IPFS:** To store data securely and in a decentralized manner, the platform utilizes the InterPlanetary File System (IPFS). This guarantees that user-submitted data is stored off-chain in a tamper-proof, accessible, and resilient manner against data loss.
- **Testing and Deployment:** Hardhat is used for the development, testing, and deployment of the Smart Contract. This Ethereum development environment enables the simulation of contract execution, the running of tests, and the deployment of smart contracts to the Ethereum network with ease and reliability.

By integrating these technologies, our platform ensures that data collection, validation, and reward processes are carried out in a secure, efficient, and transparent manner, empowering users to participate in decentralized crowdsensing campaigns with confidence.

5 Another Version of the Project

In the initial version of our project, we chose not to store or transmit the encryption key via the Smart Contract or blockchain transactions to avoid significant security risks. If the encryption key were stored on the Smart Contract, it would be publicly accessible to all network participants, potentially compromising the confidentiality of the encrypted data. Similarly, transmitting the key through transactions would expose it to interception. To address these concerns, the encryption and decryption key was hardcoded into the code and never shared between users or with the Smart Contract.

In the second version of our project, we improved the key management process to enhance both security and flexibility. We assume that the Admin holds key pair (K_{Enc}, K_{Dec}) stored locally, used to encrypt and decrypt the data.

5.1 Version 2.a

File Upload Process

1. The User that wants to upload a file generates a public-private key pair (PK_U, SK_U).
2. The User notifies the SC of the intention to send a file by transmitting the PK_U to the SC.
3. The SC generates a "UserEnrolled" event, which the Admin listens for.
4. The Admin retrieves the PK_U from the SC.
5. The Admin encrypts the key K_{Enc} using the PK_U of the User, resulting in $K_{\text{ciphered}} = \text{Enc}(K_{\text{Enc}}, PK_U)$.
6. The Admin sends the K_{ciphered} to the SC.
7. The User retrieves K_{ciphered} from the SC and decrypts it using its SK_U to obtain $K_{\text{Enc}} = \text{Dec}(K_{\text{ciphered}}, SK_U)$.
8. The User encrypts the data with K_{Enc} and uploads it to IPFS.
9. The User communicates the IPFS data location to the SC.

Validation Process

1. The SC generates a verification request event, which is monitored by the Verifier.
2. The Verifier contacts the SC to register its intent to verify, sending its public key (PK_V).
3. The SC generates a "VerifierEnrolled" event, which is monitored by the Admin.
4. The Admin retrieves Verifier's public key (PK_V) from the SC.
5. The Admin encrypts K_{Dec} using PK_V, resulting in $K_{\text{ciphered}} = \text{Enc}(K_{\text{Dec}}, PK_V)$.
6. The Admin sends K_{ciphered} to the SC.
7. The Verifier retrieves K_{ciphered} and the file's CID from the SC.
8. The Verifier decrypts K_{ciphered} using its private key (SK_V) to obtain K_{Dec} = $\text{Dec}(K_{\text{ciphered}}, SK_V)$.
9. The Verifier fetches the data from IPFS using the CID.
10. The Verifier decrypts the data using K_{Dec}.
11. The Verifier verifies the data and sends the response to the SC.

6 Gas Usage Analysis

An analysis was carried out regarding gas consumption during transactions on the blockchain. Gas is a unit that measures the computational effort required to execute operations on the Ethereum network, such as deploying smart contracts, executing functions, or storing data. Each operation in a transaction has an associated gas cost based on its complexity and the resources it consumes, including processing power and storage space.

In the case of our DApp, gas consumption occurs during various stages of the transaction process. For example, when a user uploads data, gas is consumed for encrypting the file, uploading it to IPFS, and recording the IPFS hash on the blockchain, each of which requires computational effort. Similarly, the verification process incurs gas costs when the verifier retrieves the encrypted data, checks its integrity, and decrypts it using the appropriate key.

By carefully analyzing gas consumption, we can ensure that the platform is efficient and economically viable while maintaining fairness and transparency. Understanding these costs allows for the fine-tuning of the system's fee structure to ensure users are not overcharged and verifiers are adequately rewarded, ultimately leading to a balanced and sustainable DApp ecosystem.

6.1 Gas consumption

The analysis of gas consumption reveals that the deployment of the smart contract—being a complex process with significant storage requirements—consumes 3,847,627 gas out of the 30,000,000 gas limit. For data uploads, the gas cost for the first upload is 168,178, reflecting the effort to securely store the reference to the uploaded data, while subsequent uploads require only 151,078 gas. Similarly, the verification process initially consumes 99,310 gas out of a limit of 104,741, as it involves ensuring data authenticity and integrity, whereas subsequent verifications are more efficient, requiring only 82,210 gas.

Process	**Gas Consumption**	**Gas Limit**
Smart Contract Deployment	3,847,615	30,000,000
Data Upload (First Time)	168,178	168,178
Data Upload	151,078	151,078
Data Verification (First Time)	99,310	104,741
Data Verification	82,210	87,027

Table 1: Gas Consumption During Transaction Process

6.2 Gas consumption in Version 2a

In the second version of our project, the enhanced key management process introduces additional cryptographic and event-handling operations, increasing gas consumption. Specifically, the steps involving the generation of public-private key pairs, the transmission of the public key to the Smart Contract, and the subsequent encryption and decryption of the key pairs (K_{Enc}, K_{Dec}) are new sources of gas consumption. Additionally, the Smart Contract must handle the transmission of the encrypted key and the generation of new events such as "UserEnrolled" and "VerifierEnrolled", which the Admin listens for. Each of these operations requires computational resources, contributing to the overall gas cost.

The gas consumption analysis below quantifies the costs associated with these security enhancements, providing insights into the trade-offs between cryptographic robustness and transaction efficiency.

Process	**Gas Consumption**	**Gas Limit**
Smart Contract Deployment	4,351,407	30,000,000
User: Upload Public Key (First Time)	162,124	162,124
User: Upload Public Key	57,032	57,032
Admin: Send Encrypted K_{Enc} (First Time)	462,492	462,492
Admin: Send Encrypted K_{Enc}	135,100	135,100
Data Upload (First Time)	168,222	168,222
Data Upload	151,122	151,122
Verifier: Verification Request	41,790	41,790
Verifier: Upload Public Key (First Time)	162,211	162,211
Verifier: Upload Public Key	57,119	57,119
Admin: Send Encrypted K_{Dec} (First Time)	1,638,008	1,638,008
Admin: Send Encrypted K_{Dec}	438,516	438,516
Data Verification (First Time)	99,332	104,764
Data Verification	82,232	87,050

Table 2: Gas Consumption in Version 2a