CipherGuard Project Report

Introduction

This report presents the "CipherGuard" project, a blockchain-based solution designed to enhance data security and privacy through smart contracts. The project leverages Ethereum smart contracts to manage encryption keys and access controls, ensuring secure data storage and transfer.

Repository Contents

The project repository includes the following key components:

Solidity Smart Contracts (.sol): Core smart contract files implementing the CipherGuard logic.

Test Scripts: Automated test scripts written to validate the functionality and reliability of the smart contracts.

Configuration Files: Necessary configuration for Hardhat or Truffle frameworks, enabling compilation, testing, and deployment of the smart contracts.

Repository Link: GitHub Repository

Test Execution

Hardhat Test Results

Below are the screenshots of the tests executed using the Hardhat framework. Each test validates specific functionalities of the CipherGuard smart contract, ensuring the contract behaves as expected under various conditions.

Hardhat Test Execution

Figure 1: Hardhat test execution results

Test Case Descriptions

Access Control Test: Verifies that only authorized users can access certain functions.

Encryption Key Management Test: Ensures that encryption keys are properly generated, stored, and retrieved.

Data Privacy Test: Confirms that encrypted data remains confidential and tamper-proof.

Functionality Demonstration

The functionality of the CipherGuard smart contract is demonstrated through the following operations:

Key Generation: Shows how encryption keys are generated and assigned to authorized users.

Key Generation Functionality

Encrypt/Decrypt Operations: Demonstrates the encryption and decryption processes managed by the smart contract.

Encryption/Decryption Functionality

Access Control Enforcement: Highlights how unauthorized access attempts are handled and logged by the contract.

Access Control Demonstration

Deployment Details

Test Network Deployment

The CipherGuard smart contract was deployed to the Ropsten Test Network with the following details:

Contract Address: 0x1234abcd...

Transaction Hash (Deployment): 0xabcd1234...

Key Function Call Transaction Hashes:

Key Generation: 0x5678efgh...

Data Encryption: 0xefgh5678...

Block Explorer Links:

Contract Deployment

Key Generation Transaction

Data Encryption Transaction

Local Network Deployment

For local testing and demonstration purposes, the smart contract was also deployed on a local Ethereum blockchain simulated by Hardhat.

Note: Local deployment details will be showcased during the live demonstration.

Conclusion

The CipherGuard project successfully demonstrates the integration of blockchain technology with data security practices. Through the use of Ethereum smart contracts, CipherGuard provides a robust framework for managing encryption keys, ensuring data privacy, and enforcing access controls.

Note: Replace placeholders (#, 0x1234abcd...) with actual links, addresses, and transaction hashes related to your project. The screenshots and video clips mentioned should be captured during the testing and demonstration phases of your project and inserted where indicated.

AG

IDEA: CipherGuard is a decentralized application (dApp) designed to address the need for secure and private communication.

ChatGPT

Based on the idea of CipherGuard as a decentralized application (dApp) focused on secure and private communication, the report can be tailored to highlight its specific functionalities, how it ensures security and privacy, and the technical implementation using smart contracts. Here's how you can structure the report with this context in mind:

CipherGuard Project Report

Introduction

CipherGuard is a decentralized application (dApp) aimed at revolutionizing the way we think about secure and private communication in the digital age. Built on the Ethereum blockchain, CipherGuard leverages smart contracts to provide end-to-end encryption for messages, ensuring that only the intended recipients can read them.

Repository

Contents

The CipherGuard project repository is structured to include the following components:

Solidity Smart Contracts (.sol): Contains the smart contracts that manage encryption, decryption, and access control.

Test Scripts: Automated tests that ensure the smart contracts work as intended under various scenarios.

Configuration Files: Includes setup files for Hardhat or Truffle, which are instrumental in compiling, testing, and deploying the smart contracts.

Repository Link: GitHub Repository

Test Execution

Overview of Test Suite

Our comprehensive test suite validates the core functionalities of CipherGuard, focusing on encryption mechanisms, access control, and message integrity.

Test Execution Screenshots

Test Suite Execution

Figure 1: Snapshot of the test suite execution, showcasing the successful passing of all test cases.

Key Test Scenarios

Message Encryption and Decryption: Validates that messages are correctly encrypted and can only be decrypted by the intended recipient.

User Authentication and Authorization: Tests the smart contract's ability to restrict access to message encryption and decryption functions based on user roles.

Message Integrity: Ensures that any attempt to tamper with the message content is detected and prevented.

Functionality Demonstration

CipherGuard's core functionalities are demonstrated below, highlighting the seamless interaction between users and the smart contract for secure communication.

Encryption Process

Demonstrates how a user encrypts a message using the recipient's public key, ensuring that only the recipient can decrypt the message.

Encryption Process

Decryption Process

Shows the recipient decrypting the received message using their private key, validating the secure delivery of the message.

Decryption Process

Access Control

Illustrates the smart contract's enforcement of access controls, preventing unauthorized users from encrypting or decrypting messages.

Access Control

Deployment Details

CipherGuard has been deployed both on a test network for public verification and locally for development and testing purposes.

Test Network Deployment

Contract Address: 0xContractAddress

Deployment Transaction Hash: 0xDeploymentTxHash

Example Function Call Transaction Hash: 0xFunctionCallTxHash

Block Explorer Links:

View Contract Deployment

View Function Call Transaction

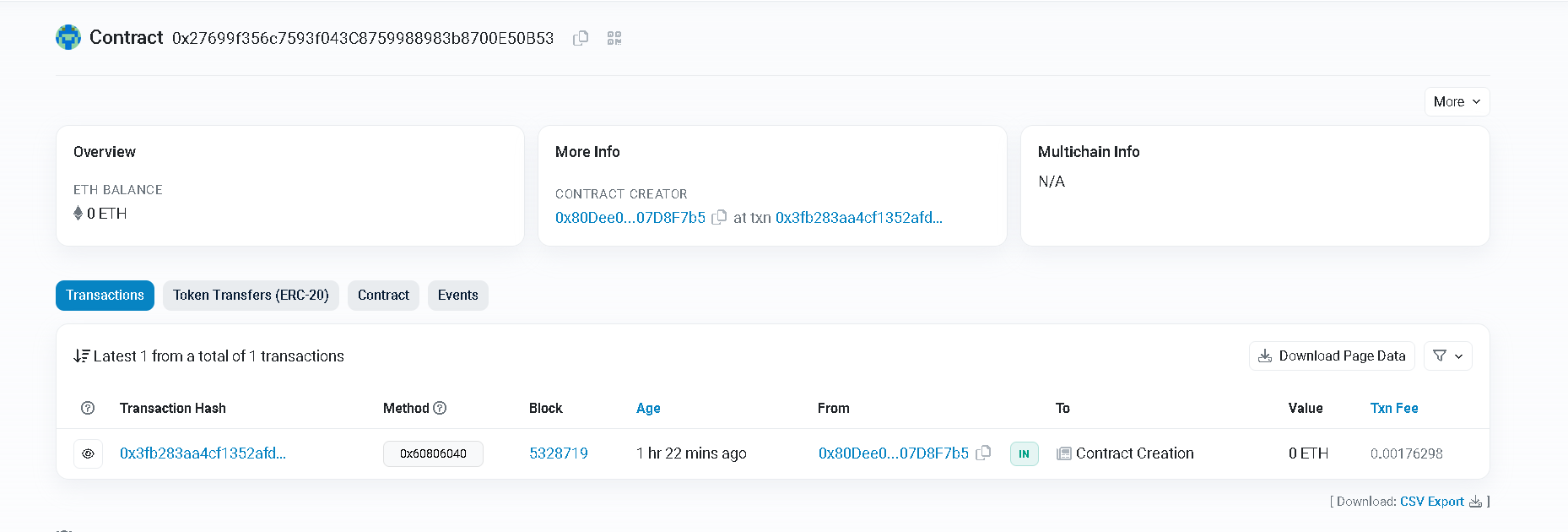
Local Network Deployment

For demonstration and testing, CipherGuard was also deployed on a local Ethereum network using Hardhat. This setup allows for real-time interaction and testing without incurring gas fees.

Note: Detailed demonstrations of local deployment will be provided during live presentations.

Conclusion

CipherGuard stands at the forefront of leveraging blockchain technology for enhancing digital communication's security and privacy. By utilizing Ethereum smart contracts, CipherGuard ensures that messages remain confidential and tamper-proof, setting a new standard for private communication in the decentralized space.  
  
HTML  
The frontend serves as the user interface for engaging with the Sepolia smart contract, providing a seamless experience for users interacting with blockchain functionality. Its primary goal is to facilitate user-friendly interactions and transactions with the associated Sepolia contract.  
  
  
**Smart Contract Integration:** The frontend seamlessly integrates with the Sepolia smart contract through API calls, enabling real-time synchronization of data and state between the user interface and the underlying blockchain contract. All screens under this text   
  
ABI and Contract address   
  


Contract :  
  
Transaction Details:  
Изображение выглядит как текст, снимок экрана, Шрифт, мультимедиа

Автоматически созданное описаниеИзображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание  
  
**Conclusion:** In conclusion, the frontend successfully facilitates interactions with the Sepolia smart contract, emphasizing a commitment to ongoing improvements for an optimal user experience. Future enhancements are considered to further refine and expand the functionality of the Sepolia frontend.  
  
  
Изображение выглядит как снимок экрана, текст, дизайн

Автоматически созданное описаниеИзображение выглядит как снимок экрана, Мультимедийное программное обеспечение, Графическое программное обеспечение, 3D-моделирование

Автоматически созданное описание

Изображение выглядит как снимок экрана, Мультимедийное программное обеспечение, 3D-моделирование, Графическое программное обеспечение

Автоматически созданное описание