Implementing a Secure Messaging System on the Polygon Blockchain

João André Gomes Marques
Department of Informatic Engineering, University of Coimbra
uc2017225818@student.uc.pt

Leonardo Oliveira Pereira
Department of Informatics Engineering, University of Coimbra
uc2020239125@student.uc.pt

Abstract

This paper presents a theoretical framework for a secure messaging system on the Polygon blockchain, focusing on the integration of SOAP protocol, AES and RSA encryption. While it primarily lays the groundwork without actual implementation, the study details the initial steps of project planning, smart contract development, and deployment on the Polygon Mumbai testnet. The research provides insights for developers in the field, outlining key considerations, best practices, and potential challenges in developing secure messaging applications on blockchain platforms.

Index Terms

Blockchain Technology, Secure Messaging, Polygon Blockchain, Smart Contracts, SOAP Protocol, RESTful APIs, AES Encryption, RSA Encryption, Hash Functions

I. Introduction

Blockchain technology is emerging as a tool across various industries. As a decentralized and immutable ledger, blockchain has been instrumental in transforming financial transactions, supply chain management, digital rights in the music industry, and government processes through smart contracts. This research delves into the application of blockchain for enhancing secure messaging systems, with a specific focus on the Polygon blockchain, a Layer 2 solution that extends the capabilities of Ethereum [1].

Traditional messaging applications, while widely used, face inherent challenges in handling large-scale communications, ensuring robust security, and providing advanced features like group video calls. These limitations are particularly pronounced in centralized systems where data encryption and security vulnerabilities remain a concern. Blockchain technology, with its decentralized and immutable nature, offers a promising alternative. This study introduces a novel messaging solution designed to leverage the unique attributes of blockchain, such as decentralized control, immutable data storage, and enhanced security protocols. While acknowledging the advancements in existing blockchain messengers like Status and Session, the research also identifies critical challenges in scalability, user adoption, and compliance with legal frameworks [2].

A. Problem Statement

This paper addresses the development of a secure messaging system on the Polygon blockchain. It identifies the necessity for decentralized messaging solutions in light of the scalability, user adoption, and legal compliance challenges faced by existing secure messaging applications. Integrating Push Protocol with Polygon's PoS chain is a step in this direction, aiming to provide a secure, user-centric, and private messaging framework in a decentralized context [1].

B. Research Objectives

The study sets several objectives: ensuring message integrity and non-repudiation against unauthorized alterations, implementing robust user authentication leveraging blockchain's cryptographic strength, developing privacy-centric messaging features, and exploiting Polygon's Layer 2 scaling solutions for efficient transaction handling.

The subsequent sections of the paper will detail the technical implementation, comparative analyses with existing systems, and potential future enhancements in this rapidly evolving field.

II. Literature Review

A. Blockchain and Smart Contracts

Blockchain, a distributed ledger technology, has been transforming secure communication by providing a decentralized and immutable platform for transaction recording [3]. A significant advancement within this domain is the emergence of smart contracts – self-executing contracts with terms directly written into code. These contracts, when deployed on blockchains like Polygon, are crucial for developing reliable applications.

The shift towards blockchain in messaging systems indicates a transition from the limitations of traditional centralized methods. Despite using secure protocols such as end-to-end encryption, these systems frequently faced challenges related to centralized governance, potentially compromising user privacy and autonomy. Highlighting these vulnerabilities, recent advancements like Quarks have emerged, utilizing Distributed Ledger Technology (DLT) to decentralize control and enhance security, thereby addressing the critical flaws in previous systems [4]. This innovation marks a significant leap in secure messaging, with preliminary testing indicating its potential scalability and effectiveness in practical applications. It addresses past limitations of centralized systems by providing a decentralized, tamper-proof platform for communication and transactional exchanges. This integration not only enhances message security but also assures the reliability of transactional processes within the messaging system.

B. Polygon Overview

Polygon stands an a Layer 2 scaling solution, enhancing the Ethereum blockchain's transaction speed and cost efficiency [5]. This platform is integral to the development of scalable decentralized applications (DApps), notably in the domain of messaging systems. Polygon's architecture empowers developers to create DApps using Ethereum's development tools, bridging the best aspects of Ethereum and sovereign blockchains.

Gwyneth Iredale's article [6] highlights Polygon's significance within the Ethereum ecosystem, particularly its evolution into an interoperable blockchain scaling framework. This transformation underscores Polygon's role in resolving the scalability issues that have long challenged global blockchain networks, directly impacting the efficiency of DApps.

In addressing scalability, Polygon integrates a spectrum of technologies, including Proof-of-Stake, Zk Rollups, and Optimistic Rollups, to elevate transaction throughput. Such advancements are crucial for the performance of blockchain-based systems, especially messaging applications. Furthermore, Polygon's commitment to the Ethereum ecosystem is evident in its adoption of plasma scaling technology and alignment with Ethereum Improvement Proposals (EIPs). The platform's security layer, employing the concept of "validators as a service," ensures the integrity of its blockchain network [6].

Polygon's unique approach to scalability, coupled with its integration with Ethereum, positions it as a key player in enhancing network performance and user experience for decentralized applications [7].

C. Solidity Programming

Solidity, an object-oriented programming language for Ethereum, is used for developing smart contracts. Its syntax, influenced by C++, JavaScript, and Python, makes it accessible and versatile for various programming backgrounds [8].

The language's integration with the Ethereum Virtual Machine (EVM) ensures compatibility and efficient execution of smart contracts. This is essential for developing sophisticated features in decentralized messaging applications, such as automated transactions and privacy controls. Solidity's static typing and contract-oriented approach contribute to the reliability and security of these applications, facilitating a decentralized, robust messaging environment [9].

Blockchain-based messaging platforms such as Whisper, Matrix, and Status, as highlighted by Sadiq [10], utilize Solidity for implementing features like encrypted communication and seamless transaction integration within messages. Sadiq's work provides an in-depth exploration of Solidity, covering aspects from syntax to smart contract deployment, underscoring its prominence in blockchain development. Despite alternatives like Vyper, Solidity's extensive community support and high demand in the market make it a preferred choice for developers.

Solidity's significance in blockchain technology, especially for secure and efficient smart contract implementation in messaging systems, is indispensable. It offers the necessary tools for developers to create scalable, secure decentralized applications within the Ethereum ecosystem.

D. SOAP Protocol

SOAP (Simple Object Access Protocol), a fundamental protocol in web services, enables communication across different operating systems using HTTP and XML [11]. Its development was marked by the need to balance security and operational efficiency, especially given its ability to traverse firewalls, posing potential security risks.

Studies, like those by Chakaravarthi et al., have highlighted the security concerns in SOAP message transactions, particularly in the context of user authentication and web service security [12]. The emergence of Web Services Security (WSS) marked a significant advance in addressing these vulnerabilities, focusing on secure message exchanges without compromising performance.

The advent of blockchain technology has introduced new considerations in cybersecurity, particularly for SOAP-based systems. The unique features of blockchain, such as decentralization and immutability, complement SOAP's security measures. Research by Bennett suggests exploring the integration of blockchain's robust security mechanisms with SOAP to enhance data transmission security and resilience against evolving cyber threats [13].

E. Current State of Blockchain-Based Messaging Systems

Recent advancements in blockchain-based messaging systems demonstrate a diverse range of approaches. Sartekin et al. and Mirzaei et al. utilized the Interplanetary File System (IPFS) in their applications 'CrypTouch' and 'Simorgh', respectively, for off-chain message storage [14], [15]. However, these systems encounter challenges in handling smaller messages and necessitate significant resources for managing the entire messaging system.

Abdulaziz et al. proposed a chat application using Ethereum's Whisper protocol, aimed at secure, anonymous communication [16]. Despite its privacy advantages, this approach suffers from high processing and memory demands, making it less viable for efficient messaging. Menegay et al. explored a communication platform on the 'Steem' blockchain, but the public content nature of 'Steem' limits its suitability for private communication applications.

Centralized control in messaging systems, as discussed by Sharma et al. and prevalent in platforms like Signal [17], WhatsApp [18], and others [19], [20], [21], poses significant security risks and potential points of failure. Rösler et al. identified critical security gaps in the group chat features of Signal, WhatsApp, and Threema, particularly in ensuring confidentiality and future secrecy [22].

Blockchain-based email systems such as LedgerMail [23], CryptaMail [24], and Swiftmail [25] record transactions on the blockchain, ensuring data permanence but raising concerns about long-term data storage and vulnerability to attacks. Invisible Ink offers a solution for storing email hashes on the Bitcoin blockchain while allowing deletion through a separate database, but it complicates credential management and operates on a public blockchain [26].

Decentralized public key infrastructure (PKI) solutions explored by Yakubov et al., Axon and Goldsmith, and Lewison and Corella emphasize user control over public key management, addressing key concerns in secure messaging [27], [28], [29]. These studies collectively highlight the pressing need for a decentralized messaging system that enhances user privacy, security, and data control.

This analysis underscores the challenges and evolving needs in blockchain-based messaging systems, pointing towards the necessity for innovative solutions that prioritize user-centric privacy and security in decentralized communications.

III. Polygon vs Other Blockchains: A Comparative Analysis

Polygon's distinct advantages make it a suitable choice for blockchain-based messaging systems:

- Scalability: Polygon addresses Ethereum's scalability issues with Layer 2 solutions like sidechains, boosting transaction throughput essential for messaging systems. It achieves up to 65,000 tps with block confirmation times of 2 seconds, catering to high-volume communication needs [30].
- Low Transaction Costs: Compared to Ethereum, Polygon offers significantly reduced transaction fees, averaging around $0.01. This is particularly beneficial for messaging applications involving frequent small transactions [31].
- Interoperability and EVM Compatibility: Polygon ensures seamless integration with Ethereum and other blockchains, offering a broad scope for application development and cross-chain interactions.
- Enhanced User Experience: The network's design minimizes latency and accelerates transaction confirmations, crucial for efficient and user-friendly messaging applications.
- Smart Contract Capabilities: Supporting advanced smart contract functionalities, Polygon enables features like end-to-end encryption and token-based rewards within messaging platforms.
- Community and Ecosystem Support: A strong community and ecosystem around Polygon provide developers with resources and collaborative opportunities for building innovative messaging solutions.
- Sustainability: The adoption of proof-of-stake consensus reduces environmental impact, aligning with sustainable technology goals.

We chose Polygon as the foundation for our study due to its standout attributes. Polygon offers interoperability and efficient consensus with a 2-second block processing time. It scales to support 65,000 transactions per second, is compatible with popular smart contract languages like Solidity and Vyper, and seamlessly integrates with the Ethereum Virtual Machine (EVM) ecosystem. In the next section, we provide further details about this blockchain.

A. Limitations

Despite its advantages, Polygon faces certain limitations:

- Dependency on Ethereum: Polygon's reliance on the Ethereum mainchain for key operations presents vulnerabilities. Any disturbances, security issues, or significant changes in Ethereum could directly impact Polygon's stability and functionality. This interdependence can lead to cascading effects within the Polygon ecosystem in the event of Ethereum's network congestion or security incidents.
- Centralization Concerns: While Polygon aims to decentralize, it faces criticism over its level of decentralization. The concern primarily revolves around the control exerted by a limited number of validators. These validators, significant in the block creation and voting process, could potentially lead to centralization if influenced or owned by a few entities.

Moreover, in the PoS mechanism, entities with substantial staked tokens could influence the network's governance. Addressing these issues requires proactive measures by the Polygon community to promote wider participation in staking and governance and to implement safeguards against centralized control.

These limitations highlight essential areas for Polygon's ongoing development, focusing on reducing Ethereum dependence and enhancing decentralization to ensure a more resilient and autonomous ecosystem.

B. Blockchain Messaging Systems

A comparison with other messaging platforms reveals varied approaches and unique features for these systems:

- Solana (Secretum): Known for high processing capabilities and low transaction fees, Solana employs a Proof-of-History (PoH) and Proof-of-Stake (PoS) consensus mechanism, enhancing scalability and efficiency for messaging applications. Its security is bolstered through advanced consensus methods and cryptography.
- Sense Chat: Utilizing the EOS blockchain, Sense Chat leverages a decentralized structure to enhance security and resist censorship. EOS's high transaction processing capability and developer-friendly environment make it a suitable choice for messaging applications [32].
- Vibeo (VBEO): Operating on the Ethereum blockchain, Vibeo integrates messaging with additional functionalities like video conferencing and location sharing. It uses blockchain technology for secure messaging and incorporates VBEO tokens for in-app transactions.
- Dust (DUST): Dust offers a high level of privacy through encryption and auto-deletion of messages after 24 hours, aligning with the blockchain's emphasis on privacy.
- Kik (Kin): Originally using Ethereum for its Kin cryptocurrency, Kik transitioned to the Solana blockchain, offering users new ways to earn revenue within the app.
- Kakao Talk: A South Korean messaging app with video call capabilities, Kakao Talk is also exploring blockchain technology for new developments.
- Status: Built on the Ethereum blockchain, Status offers messaging, a web wallet, and access to Ethereum's DApps, allowing encrypted communications and transactions with Ether (ETH).

Blockchain-based messaging apps represent a significant shift in digital communication, offering a range of features that extend beyond traditional messaging applications. The use of digital wallets, decentralized infrastructures, and user empowerment are key differentiators. As blockchain technology continues to evolve, it is poised to challenge the dominance of conventional messaging platforms, further transforming the landscape of digital communication and beyond.

IV. Message System Design

Our messaging system design prioritizes security and usability with the following key features:

A. User Authentication

Ensuring secure access, only registered users can send and receive messages. Strong authentication methods using cryptographic protocols and digital signatures restrict system access to authorized individuals. This approach is integral to maintaining secure communication, as proposed in the Polygon 2.0 framework.

B. Data Integrity

We utilize cryptographic principles to ensure message data remains unaltered, leveraging Polygon 2.0's Proving Layer. This layer, with its ZK proving protocol, guarantees the integrity of each message transaction across the network [33].

C. Message Privacy

Encryption is employed to ensure message confidentiality, with the Interop Layer of Polygon 2.0 facilitating secure cross-chain messaging. This encryption ensures that messages are readable only by the intended recipients.

SOAP Protocol Integration: Messages are exchanged using the SOAP protocol over HTTPS, enhancing secure communication. The use of SOAP over HTTPS ensures reliable and encrypted message exchange, aligning with standard security practices.

RESTful Message Capture: The system employs RESTful services for efficient message exchange, contributing to its scalability and adaptability. RESTful architecture, as demonstrated in applications like the Polygon.io Crypto API, supports diverse message formats and facilitates seamless integration.

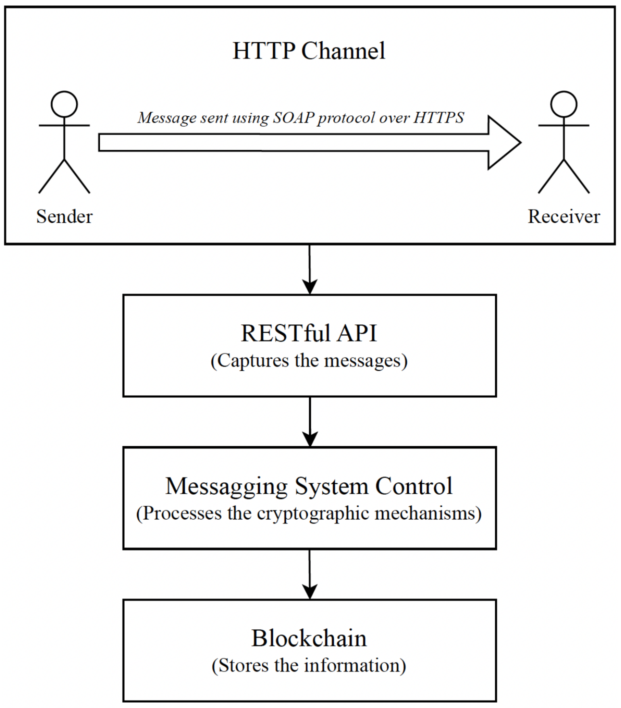
HTTP Channel
Message sent using SOAP protocol over HTTPS
Sender
Receiver
RESTful API
(Captures the messages)
Messagging System Control
(Processes the cryptographic mechanisms)
Blockchain
(Stores the information)

Fig. 1: Messaging System Architecture

The design of our messaging system integrates user authentication, data integrity protocols, and encrypted communication channels. The incorporation of SOAP and RESTful services further enhances its security, reliability, and scalability.

V. Secure Message Exchange and Encryption

Our messaging system emphasizes the critical role of cryptographic principles in user interaction, message transmission, and retrieval. It integrates advanced cryptographic techniques, ensuring secure communication, data integrity, and non-repudiation, particularly in the face of vulnerabilities such as Man-in-the-Middle (MitM) attacks.

A. The Need for Advanced Encryption beyond HTTPS

The system uses the SOAP protocol over HTTPS utilizing a client-server model where the server-side SOAP processes smart contract functions. While HTTPS provides a basic level of message encryption and security, this alone may not be adequate for resolving disputes effectively. In scenarios like MitM attacks, where communication between two systems is intercepted and potentially altered, the integrity of the message can be compromised.

The system's architecture integrates cryptographic elements for securing user identity, maintaining message confidentiality, and preserving data integrity on the Polygon network. This approach establishes a resilient, privacy-enhancing messaging platform.

B. Encryption Methods

We employ two primary encryption methods: RSA for authentication and non-repudiation, and AES for encrypting the message body. This dual approach ensures that only intended recipients can decrypt messages, thus enhancing message confidentiality.

1) RSA (Rivest-Shamir-Adleman) Encryption with Hash Functions:

- Initially, the hash function SHA-256 processes the message to generate a hash value.
- The sender encrypts this hash value with their private RSA key, creating a digital signature.
- Upon receiving the message and its signature, the recipient decrypts the signature with the sender's public key to retrieve the hash value. They then hash the received message themselves.
- To ensure a non-repudiable communication, the recipient encrypts this newly calculated hash with their private key.
- The system then verifies this encrypted hash against the sender's original hash to confirm message integrity and authenticity and allows the read the body message.

2) AES (Advanced Encryption Standard) Encryption):

- The sender and recipient agree on a shared secret AES-256 key.
- The sender encrypts the body message with this key before transmission.
- Upon receipt, the recipient uses the same key to decrypt the message.

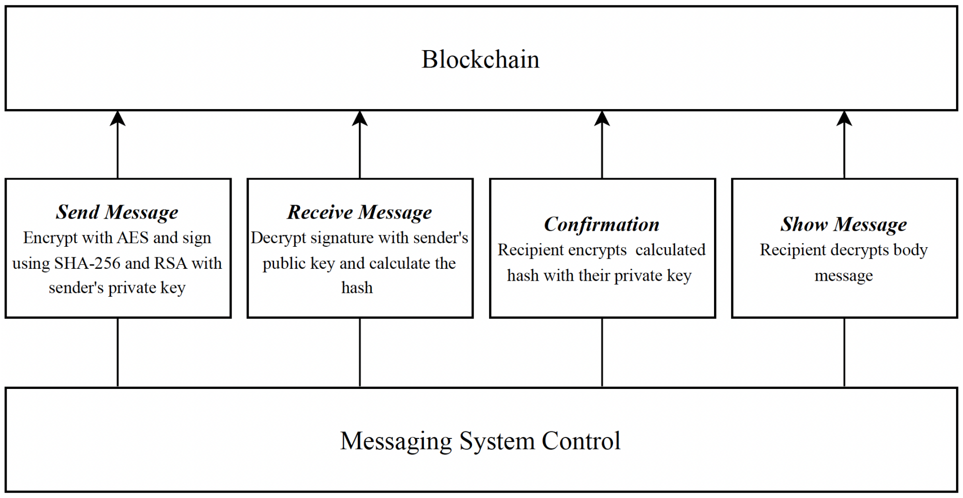
Blockchain
Send Message
Encrypt with AES and sign using SHA-256 and RSA with sender's private key
Receive Message
Decrypt signature with sender's public key and calculate the hash
Confirmation
Recipient encrypts calculated hash with their private key
Show Message
Recipient decrypts body message
Messaging System Control

Fig. 2: Cryptographic Architecture

VI. Development Environment Setup

For the development of a secure messaging system on Polygon, setting up an appropriate environment is crucial:

1. Setting up Truffle Suite

- Install Truffle Suite globally using npm: `npm install -g truffle`
- Truffle Suite offers a comprehensive framework for developing, testing, and deploying smart contracts and dApps. It provides a development environment, testing framework, and asset pipeline tailored for Ethereum and Ethereum-compatible networks like Polygon [34], [35].

2. Setting up Ganache

- Install Ganache CLI globally with npm: `npm install -g ganache-cli`
- Ganache serves as a private Ethereum blockchain, offering a controlled environment for smart contract development and testing. It facilitates the deployment of contracts, application development, and running tests in an isolated environment [36].

Truffle Suite and Ganache together create a robust development environment for the secure messaging system on Polygon. This setup not only streamlines the development process but also supports the system's security, making the development and deployment of smart contracts more efficient and reliable [37]. The choice of these tools aligns with the project's aim to develop a secure and dependable communication network on the Polygon Blockchain.

VII. Project Initialization

Initiating the project for a secure messaging system on Polygon involves setting up the development environment using Truffle Suite:

1) Creating the Project Directory: Execute `mkdir SecureMsgDapp && cd SecureMsgDapp` to create and switch to the "SecureMsgDapp" directory. This command establishes the workspace for the project.
2) Initializing the Truffle Project: Run `truffle init` within the "SecureMsgDapp" directory. This command sets up a Truffle project, creating the necessary structure including the "truffle-config.js" file, migration scripts, sample contracts, and a testing directory. This structured setup is crucial for the streamlined development and testing of smart contracts [37].

These steps ensure a well-organized development environment, facilitating focused work on building and testing the smart contracts essential for the secure messaging system. This initial setup is fundamental for effective and efficient development on blockchain platforms like Polygon.

VIII. Smart Contract Implementation

In this section, we will discuss the implementation of the smart contract for our secure messaging system. The contract is written in Solidity.

A. User Registration and Authentication

We employ a registration function and a mapping to keep track of users. The ‘onlyRegistered’ modifier ensures that a function is accessible to authenticated users only. The contract also includes a ‘User’ structure to store user information and a ‘Message’ structure to store message data:

```
pragma solidity ^0.8.0;

contract SecureMessaging {
    struct User {
        string name;
        string publicKey; // Public key for message encryption
    }

    struct Message {
        address sender;
        string content;  // This would be encrypted in a real-world scenario
        uint timestamp;
    }

    mapping(address => User) private users;
    mapping(address => Message[]) private inboxes;

    event UserRegistered(address indexed userAddress, string name, string publicKey);
    event MessageSent(address indexed from, address indexed to, string content, uint
    timestamp);

    modifier onlyRegistered {
        require(bytes(users[msg.sender].name).length != 0, "User not registered.");
        _;
    }

    // Registration function
    function registerUser(string memory _name, string memory _publicKey) public {
        require(bytes(users[msg.sender].name).length == 0, "User already registered.");
        users[msg.sender] = User(_name, _publicKey);
        emit UserRegistered(msg.sender, _name, _publicKey);
    }

    // User authentication is implicit in the fact that a message sender must be registered

    // Message sending function
    function sendMessage(address _recipient, string memory _content) public onlyRegistered
    {
        require(bytes(users[_recipient].name).length != 0, "Recipient not registered.");
        Message memory newMessage = Message(msg.sender, _content, block.timestamp);
        inboxes[_recipient].push(newMessage);
        emit MessageSent(msg.sender, _recipient, _content, block.timestamp);
    }

    // Message retrieval function
    function getMyMessages() public view onlyRegistered returns (Message[] memory) {
        return inboxes[msg.sender];
    }

    // Additional functions for user and message management can be added here
}
```

The smart contract ‘SecureMessaging’ is designed to facilitate user authentication, secure message exchange, and user registration on the Ethereum blockchain, focusing on security and usability:

- Contract Structure: The smart contract begins with a Solidity version pragma for compatibility with version 0.8.0. It defines two key structures: ‘User’ for storing user details like name and public key, and ‘Message’ for encrypted message content, sender’s address, and timestamp. Private mappings, ‘users’ and ‘mail inboxes,’ are used to link user addresses with their ‘User’ structures and to store their sent messages respectively.

- Events: The contract declares two events: ‘UserRegistered’ for new user registrations capturing name, address, and public key, and ‘MessageSent’ for message transmissions, recording sender and receiver addresses, message content, and timestamp.
- Access Control: The contract employs a modifier ‘onlyRegistered’ to ensure that only registered users can access specific functionalities. It checks the length of the user’s name to verify registration status, preventing unregistered access to functions.
- User Registration: The ‘registerUser’ function allows users to register by providing their name and public key. It checks for prior registration before creating a new ‘User’ struct and emitting the ‘UserRegistered’ event.
- Message Sending: Registered users can send messages to other registered users via the ?¿sendMessage’ function, restricted by the ‘onlyRegistered’ modifier. This function creates a ‘Message’ instance, adds it to the receiver’s inbox, and triggers the ‘MessageSent’ event.
- Message Retrieval: The ‘getMyMessages’ function retrieves all messages in the sender’s inbox, utilizing the ‘onlyRegistered’ modifier for secure access to messages.
- Extensibility: The contract design allows for the inclusion of additional functions for user and message management, facilitating future expansions of the system’s capabilities.

This Solidity smart contract forms the foundation of a secure messaging system on Polygon, emphasizing user authentication, data encapsulation, and encrypted communication. It ensures secure user interactions and privacy protection, essential in blockchain applications. User registration, message encryption, and data protection are integral features, with the contract architecture facilitating a secure and authenticated messaging experience [38].

B. Deploying to Polygon

For deployment, we must configure the Polygon network in Truffle’s configuration file and write a deployment script. In ‘truffle-config.js’, we add the following configuration for the Polygon Mumbai testnet:

```
const HDWalletProvider = require('@truffle/hdwallet-provider');
const privateKey = 'YOUR_PRIVATE_KEY'; // Replace with your wallet's private key
const polygonTestnetRPC = 'https://matic-mumbai.chainstacklabs.com'; // Polygon Mumbai
    testnet RPC URL

module.exports = {
    networks: {
        mumbai: {
            provider: () => new HDWalletProvider(privateKey, polygonTestnetRPC),
            network_id: 80001,
            confirmations: 2,
            timeoutBlocks: 200,
            skipDryRun: true
        }
    }
    // Other configurations...
};
```

Deployment script:

```
const SecureMessaging = artifacts.require("SecureMessaging");

module.exports = function(deployer) {
    deployer.deploy(SecureMessaging);
};
```

These steps detail the methodical approach to deploying the ‘SecureMessaging’ smart contract, encompassing everything from initial configuration to final execution, ensuring effective deployment on the Polygon Mumbai testnet:

- Configuration in ‘truffle-config.js’: Setup for development and deployment on the Polygon Mumbai testnet, including network parameters, private keys, and RPC URLs, is defined in the “truffle-config.js” file.
- HDWalletProvider Import: The Truffle HDWallet Service package’s ‘HDWalletProvider’ is imported, enabling Truffle to sign transactions with private keys or mnemonics.
- Private Key and RPC URL Configuration: Configuration of the RPC URL and private key for the Polygon Mumbai test net, where the private key is essential for transaction signing and must be kept confidential.
- Network Configuration for Mumbai Testnet: Truffle configuration for the Mumbai testnet includes network ID, confirmation requirements, timeout blocks, and the option to skip dry runs.
- Deployment Script in ‘migrations/2_deploy_contracts.js’: Deployment of the ‘SecureMessaging’ contract is managed by the script located in the ‘migrations’ folder.

- Contract Import for Secure Messaging: The ‘SecureMessaging’ contract is imported using the ‘artifacts. require’ method in the deployment script.
- Function for Deployment: The deployment function uses Truffle’s ‘deploy’ method to deploy the ‘SecureMessaging’ contract.
- Deployment Command: The command ‘truffle migrate –network Mumbai‘ is executed to deploy the contract on the Polygon Mumbai testnet as configured in the ‘truffle-config.js’ file.

IX. Interacting with the Contract

Interacting with the ‘SecureMessaging’ smart contract deployed on the Polygon Mumbai testnet can be accomplished through the Truffle console, utilizing web3.js or ethers.js. The steps for interaction using the Truffle console are outlined below:

1) Truffle Console Setup: Open the Truffle console with the command `truffle console --network mumbai`. This connects to the Polygon Mumbai testnet.
2) Instantiating the Contract: Create a JavaScript object for the deployed contract instance using `let contract = await SecureMessaging.deployed();`.
3) Registration of Users: To register a user, invoke the contract’s ‘registerUser’ function.
Example: `await contract.registerUser(‘Alice’, ‘AlicePublicKey’);` registers ”Alice” with her public key.
4) Sending Messages: Use the ‘sendMessage’ function to send messages
For instance, `await contract.sendMessage(accounts[1], ‘Hello, Bob!’, from: accounts[0]);` sends a message from ‘accounts[0]’ to ‘accounts[1]’.
5) Retrieving Messages: Fetch messages for a user with `let messages = await contract.getMyMessages(from: accounts[1]);`, retrieving messages associated with ‘accounts[1]’.
6) Logging Messages: Display messages on the console with `console.log(messages);` to view the collected messages.

This sequence of steps provides a clear guide to interacting with the ‘SecureMessaging’ smart contract on the Polygon Mumbai testnet. It covers the setup of the Truffle console, contract instantiation, user registration, message sending and retrieval, and logging messages, demonstrating the contract’s functionalities in secure messaging.

X. Conclusion

This paper has presented a comprehensive framework for developing a secure messaging system on the Polygon blockchain. We have detailed the use of smart contracts, explored Polygon’s security features, and discussed the integration of several cryptographic techniques for robust message exchange over HTTPS. This system design prioritizes data integrity, user authentication, and message privacy in a decentralized environment.

The paper provided a step-by-step methodology for setting up the development environment, implementing smart contracts, and interacting with the deployed system. However, it is important to note the conceptual nature of this study, focusing on theoretical aspects and guidelines rather than practical implementation.

Future research should transition from theoretical constructs to practical applications and real-world testing of the proposed secure messaging app on Polygon. Potential future enhancements include decentralized identity management and the integration of artificial intelligence to enhance security. Bridging the gap between theory and practice is essential to maintain relevance in the dynamic blockchain domain.

References

[1] S. K. Rana, A. K. Rana, S. K. Rana, V. Sharma, U. K. Lilhore, O. I. Khalaf, and A. Galletta, “Decentralized model to protect digital evidence via smart contracts using layer 2 polygon blockchain,” *IEEE Access*, 2023.

[2] U. Ellewala, W. Amarasena, H. S. Lakmali, L. Senanayaka, and A. Senarathne, “Secure messaging platform based on blockchain,” in *2020 2nd International Conference on Advancements in Computing (ICAC)*, vol. 1. IEEE, 2020, pp. 317–322.

[3] GeeksforGeeks, “Blockchain Technology — GeeksforGeeks,” Accessed at 5 of January 2024. [Online]. Available: https://www.geeksforgeeks.org/blockchain-technology/

[4] M. K. Shuhan, T. Islam, E. A. Shuvo, F. H. Bappy, K. Hasan, and C. Caicedo, “Quarks: A Secure and Decentralized Blockchain-Based Messaging Network,” in *2023 IEEE 10th International Conference on Cyber Security and Cloud Computing (CSCloud)/2023 IEEE 9th International Conference on Edge Computing and Scalable Cloud (EdgeCom)*. IEEE, 2023, pp. 268–274.

[5] Polygon, “Polygon — Ethereum’s Internet of Blockchains,” Accessed at 5 of January 2024. [Online]. Available: https://polygon.technology/

[6] G. Iredale, “Polygon (MATIC) and its Importance for Ethereum,” 2021. [Online]. Available: https://101blockchains.com/polygon-matic/

[7] A. Takyar, “Polygon: Ethereum’s Internet of Blockchains,” 2020, Accessed at 5 of January 2024. [Online]. Available: https://www.leewayhertz.com/polygon-ethereums-internet-of-blockchains/

[8] Solidity, “Solidity 0.8.4 documentation,” Accessed at 5 of January 2024. [Online]. Available: https://docs.soliditylang.org/en/v0.8.4/

[9] M. Chibuzor, “Solidity Programming for Ethereum Blockchain Smart Contracts,” 2023, Accessed at 5 of January 2024. [Online]. Available: https://www.ejable.com/tech-corner/blockchain/solidity-programming-for-smart-contracts/

[10] U. Sadiq, “Role of Solidity in Blockchain Technology,” 2023, Accessed at 5 of January 2024. [Online]. Available: https://www.linkedin.com/pulse/role-solidity-blockchain-technology-umar-sadiq

[11] W3Schools, “SOAP - W3Schools,” Accessed at 5 of January 2024. [Online]. Available: https://www.w3schools.com/xml/xml_soap.asp

[12] S. Chakaravarthi, "Secure model for SOAP message communication in web services," *International Journal of Advanced Intelligence Paradigms*, vol. 8, no. 4, pp. 437–442, 2016.
[13] T. Bennett, "Understanding SOAP security," 2023, Accessed at 5 of January 2024. [Online]. Available: https://blog.dreamfactory.com/understanding-soap-security/
[14] R. A. Sarıtekin, E. Karabacak, Z. Durgay, and E. Karaarslan, "Blockchain based secure communication application proposal: Cryptouch," in *2018 6th International Symposium on Digital Forensic and Security (ISDFS)*. Ieee, 2018, pp. 1–4.
[15] E. Mirzaei and M. Hadian Dehkordi, "Simorgh, a fully decentralized blockchain-based secure communication system," *Journal of Ambient Intelligence and Humanized Computing*, vol. 13, no. 8, pp. 3903–3921, 2022.
[16] M. Abdulaziz, D. Çulha, and A. Yazici, "A decentralized application for secure messaging in a trustless environment," in *2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT)*. IEEE, 2018, pp. 1–5.
[17] "Signal," 2022, Accessed at 5 of January 2024. [Online]. Available: https://signal.org/
[18] "Whatsapp security," 2022, Accessed at 5 of January 2024. [Online]. Available: https://www.whatsapp.com/security/
[19] "Threema security," 2022, Accessed at 5 of January 2024. [Online]. Available: https://threema.ch/en/security
[20] "Open whisper systems partners with google on end-to-end encryption for allo," 2022, Accessed at 5 of January 2024. [Online]. Available: https://whispersystems.org/blog/allo/
[21] "Facebook messenger deploys signal protocol for end to end encryption," 2022, Accessed at 5 of January 2024. [Online]. Available: https://whispersystems.org/blog/facebook-messenger/
[22] P. Rösler, C. Mainka, and J. Schwenk, "More is less: On the end-to-end security of group chats in signal, whatsapp, and threema," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2018, pp. 415–429.
[23] LedgerMail, "Ledgermail home description," 2018, Accessed at 5 of January 2024. [Online]. Available: https://ledgermail.io
[24] CryptaMail, 2014, Accessed at 5 of January 2024. [Online]. Available: ttp://www.cryptamail.com
[25] NewsBTC, "Using blockchain technology for email verification," 2020, Accessed at 5 of January 2024. [Online]. Available: https://www.newsbtc.com/news/john-mcafee-swiftmail-using-blockchain-technology-for-email-verification/
[26] A. Lazarovich, "Invisible Ink: blockchain for data privacy," Ph.D. dissertation, Massachusetts Institute of Technology, 2015.
[27] A. Yakubov, W. Shbair, A. Wallbom, D. Sanda *et al.*, "A blockchain-based PKI management framework," in *The First IEEE/IFIP International Workshop on Managing and Managed by Blockchain (Man2Block) colocated with IEEE/IFIP NOMS 2018*. Tapei, Tawain, 2018, pp. 23–27.
[28] L. Axon and M. Goldsmith, "PB-PKI: A privacy-aware blockchain-based PKI," in *14th International Conference on Security and Cryptography (SECRYPT 2017)*, vol. 6. SciTePress, 2017, pp. 311–318.
[29] K. Lewison and F. Corella, "Backing rich credentials with a blockchain PKI," *Pomcor. com*, 2016.
[30] A. H. Sutopo, *Blockchain Programming Smart Contract on Polygon*. Topazart, 2023.
[31] O. Mokalusi, R. Kuriakose, and H. Vermaak, "A Comparison of Transaction Fees for Various Data Types and Data Sizes of Blockchain Smart Contracts on a Selection of Blockchain Platforms," in *ICT Systems and Sustainability: Proceedings of ICT4SD 2022*. Springer, 2022, pp. 709–718.
[32] J. Witt and M. Schoop, "Blockchain technology in e-business value chains," *Electronic Markets*, vol. 33, no. 1, pp. 1–17, 2023.
[33] A. Nieto, R. Roman, and J. Lopez, "Digital witness: Safeguarding digital evidence by using secure architectures in personal devices," *IEEE Network*, vol. 30, no. 6, pp. 34–41, 2016.
[34] R. Verma, N. Dhanda, and V. Nagar, "Application of truffle suite in a blockchain environment," in *Proceedings of Third International Conference on Computing, Communications, and Cyber-Security: IC4S 2021*. Springer, 2022, pp. 693–702.
[35] T. Suite, "Truffle — Ethereum Development Environment," Accessed at 5 of January 2024. [Online]. Available: https://www.trufflesuite.com/
[36] T.Suite, "Ganache — Personal Ethereum Blockchain," Accessed at 5 of January 2024. [Online]. Available: https://www.trufflesuite.com/ganache
[37] D. Kuonen, "The process of creating, testing, and deploying smart contracts on the ethereum blockchain using solidity," 2023.
[38] A. Vangala, A. K. Sutrala, A. K. Das, and M. Jo, "Smart contract-based blockchain-envisioned authentication scheme for smart farming," *IEEE Internet of Things Journal*, vol. 8, no. 13, pp. 10 792–10 806, 2021.