A Proxy-Layer Approach to Secure Smart Contract Deployment on Private EVM-Based PoA Blockchains

Yonghao Wang[1], Jahid Ali[2], Junaid Arshad[1], Yunxia Liu[3]
[1]College of Computing, Birmingham City University, UK
[2]MEER Labs Limited, UK
[3]Zhengzhou Normal University, China
yonghao.wang@bcu.ac.uk; jahid@meerlabs.com; junaid.arshad@bcu.ac.uk; liuyunxia0110@zznu.edu.cn

***Abstract*—Within the enterprise sector, there is a growing trend of adopting private and consortium blockchains to leverage the benefits of blockchain technology. Networks based on Proof-of-Authority (PoA) and compatible with the Ethereum Virtual Machine (EVM) are particularly advantageous in this scenario, providing both efficiency and straightforward setup. While private blockchains inherently enhance security by their permissioned nature, which restricts node participation to authorised entities only, it's crucial to recognise the ongoing necessity for additional safeguards. These safeguards are essential to guarantee that only verified and trustworthy smart contracts are deployed. A key concern in this realm is addressing particular attack vectors, such as front-running attacks. This paper presents an innovative proxy-layer solution designed to integrate effortlessly with any EVM-compatible blockchain, thereby strengthening both governance and security aspects.**

Keywords—EVM, Blockchain, PoA, Proxy, Smart Contract

I. Introduction

Since the conception and implementation of the Bitcoin blockchain in 2008 by the pseudonym Satoshi Nakamoto [1], blockchain technology has undergone significant evolution, particularly in terms of enhanced programmability [2]. This evolution has spurred widespread adoption in various industries and governmental bodies, attracted by the technology's inherent features like data transparency and immutability. However, these stakeholders are often cautious, seeking to avoid the risks associated with unregulated public blockchains, which can be susceptible to fraudulent activities such as Ponzi schemes. An illustrative example is the Chinese government's use of blockchain to securely and transparently track 'fapiao,' or certified invoices that are tax-bureau-authorised records of commercial transactions [3], utilizing a private blockchain system. As a result, many sectors are exploring the potential of consortium or private blockchains as a medium for facilitating transparent and immutable business transactions and contracts among diverse entities.

In this context, private and consortium blockchains offer a regulated, customised infrastructure that is conducive to fulfilling specific organisational objectives and regulatory compliances. These blockchains cater to a broad array of critical domains, encompassing not only regulatory compliance, data transparency, and asset tracking, but also need to fulfil special performance requirements such as speed, efficiency, enhanced security, and access control.

In this paper, we introduce a novel proxy-layer mechanism designed to enforce permissioned smart contract deployment, thereby augmenting the security framework of private blockchain implementations.

II. Background and Challenges

A. Ethical and Accessible Private Blockchain Frameworks

Within the consortium and private blockchain landscape, two dominant architectural paradigms exist: Hyperledger frameworks [4] and Ethereum-compatible blockchains using Proof of Authority (PoA) consensus mechanisms under the EIP-225 Clique specifications [5]. Hyperledger comes with modular design and enterprise-specific features, while PoA networks has advantages in terms of compatibility with the existing EVM-based decentralised applications (DApps) ecosystem and ease of setup.

Our research initiative aims to make blockchain technology accessible and ethic to Small and Medium-sized Enterprises (SMEs). Consequently, we have opted for Ethereum-compatible technologies, largely due to their mature ecosystem of pre-existing decentralised applications (DApps). This choice offers SMEs a lower cost for software development and maintenance, especially for those lacking the technical expertise or financial resources for more complex blockchain implementations. An additional advantage of Ethereum-compatible technologies is their well-established protocols for cross-chain operations with other public blockchains, that could potentially facilitate value transfer mechanisms between various types of distributed ledgers. This offers a possible value bridge between private and public chains when SMEs require such applications. While Hyperledger frameworks offer high degrees of customisation and control, they often entail intricate setup procedures that may pose barriers to SMEs, which typically lack the extensive resources of larger enterprises. In contrast, private PoA (PPOA) based frameworks within the Ethereum-compatible paradigm offer a more straightforward deployment process. This accessibility aligns well with our overarching objective to develop an ethical, transparent, and readily accessible blockchain infrastructure.

B. Security Challenges and Performance Requriements

EVM-compatible blockchains are prevalent in public networks. Adopting Private Proof of Authority (PPOA) offers ease of porting existing DApps but also imports public chain security vulnerabilities. For instance, the recent surge in Decentralized Finance (DeFi) applications has led to an increase in Front-Running attacks associated with specific smart contract categories, such as decentralized exchanges, lending protocols, and liquidation mechanisms.

In a Private Proof of Authority (PPOA) environment, where nodes are generally operated by a consortium of trustworthy business partners, risks associated with 51% [6] and Sybil attacks [7] are mitigated. However, vulnerabilities related to smart contracts, particularly Front-Running attacks, persist. This issue is exacerbated by the predictability introduced by the PoA consensus algorithms, allowing attackers to anticipate when a node will produce the next block [8]. To counter these vulnerabilities, one approach is to restrict the deployment of specific types of trading-focused smart contracts by users. Within the context of an accessible and ethical PPOA environment, such a restriction is even desirable.

Alongside this, we identify a set of key features that an accessible and ethical blockchain infrastructure should incorporate, particularly catering to SMEs, charities and potentially NGOs.

Firstly, Gas Fees: The PPOA network should be designed to incur low or even no gas fees. The intrinsic transaction fee acts as a safeguard against infinite computational loops within smart contracts. Fees should be automatically managed by the network and DApps, obviating the need for additional user intervention. This is aligned with the network's role as a business utility rather than a financial transaction platform.

Secondly, Account Security: The PPOA network should offer mechanisms, such as rekeying [9][10] or comparable technologies like smart contract based wallet [11], to allow account recovery in the event of lost private keys, whilst maintaining high security standards. This will alleviate the single point of failure associated with blockchain-related wallets.

Thirdly, Smart Contract Permissions: We propose a permissioned architecture for smart contract deployment to enhance security. An audit process for contract deployment will mitigate risks like Front-Running attacks and other malicious smart contracts, aligning with the network's objective to support internal business processes.

Fourthly, Network Bridging: A controlled bridging mechanism should be in place to facilitate secure value transfer between the PPOA network and public blockchains. This feature aims to provide a method for the secure, regulated exchange of assets without compromising the network's internal integrity.

In this work, we address the third point by employing a proxy to secure the smart contract deployment process.

III. Related Work on Permissioned Blockchain Technologies

While permissioned smart contract features are typically associated with Hyperledger architectures [12], EVM-based blockchains generally function as public chains, and thus do not inherently focus on smart contract access control and auditing. To address this, we utilise a network proxy layer to intercept JSON-RPC calls, which are commonly used for smart contract interactions with blockchain nodes. Existing projects like "ethproxy" [13] and "EthLogSpy" [14] also employ reverse proxies for capturing JSON-RPC requests, albeit for logging and debugging purposes. These intercepted requests are stored in a database and can be queried more swiftly than through a traditional blockchain explorer. In contrast, our approach employs the proxy layer specifically for smart contract permissioning.

IV. System Design and Architecture

In this section, we describe the architecture and design principles underpinning our proxy layer approach. The creation of a smart contract is accomplished by submitting state-changing transactions to the blockchain. There are two primary stateless JSON-RPC methods for this purpose: "`eth_sendTransaction`" and "`eth_sendRawTransaction`" (as outlined in the Ethereum JSON-RPC documentation). The choice between these methods depends on the location of the private key intended for signing the transaction. Our proxy is designed to intercept these JSON-RPC calls, check their content, and assess whether the Externally Owned Account (EOA) is authorised to execute this specific transaction. Access control is governed by an on-chain smart contract containing a whitelist of approved addresses.

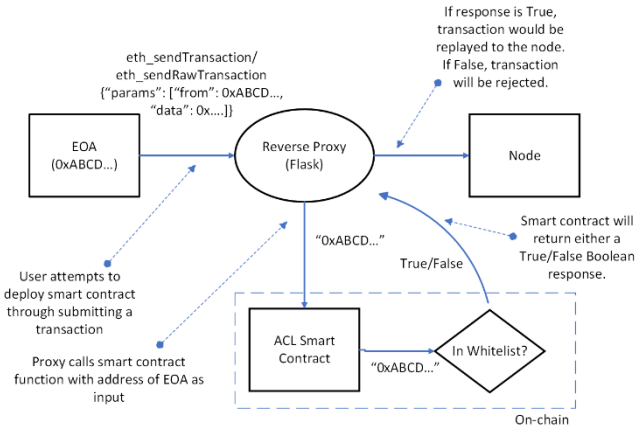

Figure 1 Proxy System Architecture

V. Implementation and Algorithmic Overview

A. Smart Contract Creation Methods

We focus on intercepting two EVM contract creation methods: 1) `eth_sendTransaction` and 2) `eth_sendRawTransaction`. The primary distinction between them lies in the handling of private keys. In the case of `eth_sendTransaction`, the node must host an open account and the associated private key to sign the contract. Whereas, `eth_sendRawTransaction` allows the client to initiate the call, ensuring the private key is neither exposed nor stored on the node. Within this context, `eth_sendTransaction` is primarily used for testing and development, while `eth_sendRawTransaction` is more suited for production environments.

Under typical conditions, the private key resides with the Externally Owned Account (EOA) intending to initiate a transaction. This transaction is constructed and locally signed by the user before being sent to the node via the `eth_sendRawTransaction` method. Unlike the prior method, these transactions are encoded using Recursive Length Prefix (RLP) [15]. They must first be decoded to extract the essential parameters needed to enforce our desired permissioned environment.

B. Smart Contract Permissioning

To check whether an EOA is attempting to deploy a smart contract, the 'params' section within either `eth_sendTransaction` or `eth_sendRawTransaction` can be examined. Specifically, the absence of a 'to' field is indicative of a contract deployment attempt. The 'from' address is then cross-referenced with a list of whitelisted addresses stored in an on-chain smart contract. If whitelisted, the transaction is permitted to pass through to the node; otherwise, it is rejected. An on-chain smart contract is utilised for access control during smart contract creation. This contract includes a function that, when invoked, returns a Boolean value (true or false) to the proxy. As the function simply reads the state of the blockchain, a response can be returned quickly without undergoing the usual process of including the transaction in a block.

C. Proxy Implementation

To build our reverse proxy, we use the Python-based Flask library, which comes with a built-in development server. Flask was chosen for its lightweight architecture and minimal boilerplate code requirements, streamlining the application development process. Another significant advantage of Flask is its extensibility, such as the Flask-CORS library, which automatically enables CORS (Cross-Origin Resource Sharing) without requiring modifications to incoming requests. This feature is particularly useful when integrating the proxy with web-based IDEs like Remix IDE.

The proxy has multiple responsibilities, including identifying the type of JSON-RPC method invoked, determining if a smart contract is being deployed, and verifying whether the user initiating the contract deployment is authorized to do so. Below is a snippet of the Python code used to monitor the `eth_sendRawTransaction` method.

Table 1 Python proxy code snippet

```
……
def eth_sendRawTransaction(signed_tx) -> bool:
      txs = rlp.decode(hex_to_bytes(signed_tx),
Transaction)
    tx_dict = txs.to_dict()

    if tx_dict["to"] == "0x":
        deployer = tx_dict["sender"]
        print(f"Deployer: {deployer}")

        return check_address(deployer)
……
```

D. Permission Contract

To build our permissioned smart contract, named `'Permission.sol,'` we employ the Solidity programming language for development and Hardhat for compilation and deployment onto an EVM-compatible test environment using Hardhat's built-in node. The contract comprises three primary functions: `'checkAddress,' 'addToWhitelist,'` and `'removeFromWhitelist.'` The `'checkAddress'` function verifies an address against a mapping called `'whiteList'` and returns a Boolean value. `'addToWhitelist'` allows new addresses to be added by setting the corresponding value to 'true,' while `'removeFromWhitelist'` revokes access by setting the value to 'false.' To secure these administrative functions against unauthorised access, we utilise the 'Ownable' smart contract from OpenZeppelin. This contract restricts specific functions to be callable only by the contract owner, enforceable by appending the `'onlyOwner'` modifier to the relevant functions.

Table 2 permissioned smart contract code snippet

```
……
pragma solidity >=0.8.2 <0.9.0;

import
"@openzeppelin/contracts/access/Ownable.sol";

contract Permission is Ownable {

    mapping (address => bool) whitelist;

     function checkAddress(address addr) public
view returns (bool) {
        bool inWhitelist = whitelist[addr];
        return inWhitelist;
    }

    function addToWhitelist(address addr) public
onlyOwner {
        whitelist[addr] = true;
    }

    function removeWhitelist(address addr) public
onlyOwner {
        whitelist[addr] = false;
    }
}
……
```

In summary, this architecture primarily consists of an on-chain permissioning smart contract and an intermediary reverse proxy. The contract regulates the access control for smart contract deployment, whilst the proxy performs operational tasks to determine whether a transaction is legitimate and permitted.

VI. Evaluation and Results

We utilise Hardhat for smart contract compilation and deployment on an EVM-compatible test environment. Furthermore, the architecture is compatible with Qitmeer's POA network, Amana, with slight modifications to the source code [16].

Two scenarios were evaluated: 1) An authorized user successfully deployed a smart contract. 2) An unauthorized user was restricted by the reverse proxy. Both scenarios produced the expected outcomes upon testing.

In addition, the architecture was deployed on Amazon Web Services (AWS) using the Qitmeer Amana Network to create a three-node private blockchain network. This involved configuration and source code modifications to Qitmeer's next-generation network (QNG). The implementation demonstrated the system's capability to effectively control and audit smart contract deployments in a production environment. The source code and instructions can be found on the relevant GitHub website [17].

In our comprehensive latency evaluation of a private Proof of Authority (PoA) testing network, we rigorously compared the contract creation delay over 100 iterations, both

with and without a proxy configuration. The results revealed a discernible difference in the network's performance in these two scenarios. On average, the network exhibited a delay of 10.96 milliseconds when operating without a proxy and a higher delay of 21.76 milliseconds with the proxy enabled. Furthermore, the variability of the network delay, as indicated by the standard deviation, was 5.86 milliseconds without a proxy and increased to 10.84 milliseconds with a proxy. These metrics are critical for understanding the impact of proxy usage on network latency within PoA blockchain environments.

VII. Conclusion and Future Directions

In this study, we introduced an intermediate proxy layer to enhance security and auditing capabilities for smart contract permissioning in private blockchain networks. Our proposed architecture only requires minor code modifications and configuration adjustments to integrate with any EVM-compatible blockchain, whether private or consortium-based. Our tests, conducted on the Qitmeer Next-Generation (QNG) network hosted on Amazon Web Services (AWS), confirmed the proxy's effectiveness in security checks.

In addition, the Python implementation of the proxy increases the average contract creation time by about 10 milliseconds, which should not be a significant barrier for user experience. With future optimisation, such as using a WebSocket proxy implementation, this figure can be further reduced.

Future work should explore securing inter-contract communications, particularly when smart contracts are generated by other contracts. Another avenue for investigation is the performance and scalability of this architecture; a thorough analysis is needed to determine its impact on block confirmation times and latency. For deployment in production environments, it is essential to secure the native RPC calls for each node within the private network. This precaution ensures the effectiveness of the auditing layer by preventing unauthorised external access.

References

[1] S. Nakamoto, ‘Bitcoin: A peer-to-peer electronic cash system’.

[2] V. Buterin and others, ‘A next-generation smart contract and decentralized application platform’.

[3] N. Kshetri, ‘Blockchains with Chinese Characteristics’, Calgary: International Telecommunications Society (ITS), 2022. Accessed: Sep. 29, 2023. [Online]. Available: https://www.econstor.eu/handle/10419/265646

[4] C. Cachin, ‘Architecture of the hyperledger blockchain fabric’, in *Workshop on distributed cryptocurrencies and consensus ledgers*, Chicago, IL, 2016, pp. 1–4. Accessed: Sep. 29, 2023. [Online]. Available: https://theblockchaintest.com/uploads/resources/IBM%20Research%20-%20Architecture%20of%20the%20Hyperledger%20Blockchain%20Fabric%20-%202016%20-%20July.pdf

[5] ‘EIP-225: Clique proof-of-authority consensus protocol’, Ethereum Improvement Proposals. Accessed: Sep. 29, 2023. [Online]. Available: https://eips.ethereum.org/EIPS/eip-225

[6] C. Ye, G. Li, H. Cai, Y. Gu, and A. Fukuda, ‘Analysis of Security in Blockchain: Case Study in 51%-Attack Detecting’, in *2018 5th International Conference on Dependable Systems and Their Applications (DSA)*, Sep. 2018, pp. 15–24. doi: 10.1109/DSA.2018.00015.

[7] T. Rajab, M. H. Manshaei, M. Dakhilalian, M. Jadliwala, and M. A. Rahman, ‘On the Feasibility of Sybil Attacks in Shard-Based Permissionless Blockchains’. arXiv, Feb. 16, 2020. doi: 10.48550/arXiv.2002.06531.

[8] Q. Wang, R. Li, Q. Wang, S. Chen, and Y. Xiang, ‘Exploring Unfairness on Proof of Authority: Order Manipulation Attacks and Remedies’, in *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, in ASIA CCS ’22. New York, NY, USA: Association for Computing Machinery, May 2022, pp. 123–137. doi: 10.1145/3488932.3517394.

[9] ‘Rekeying - Algorand Developer Portal’. Accessed: Nov. 09, 2023. [Online]. Available: https://developer.algorand.org/docs/get-details/accounts/rekey/

[10] M. Davison, K. King, and T. Miller, ‘Blockin: Multi-Chain Sign-In Standard with Micro-Authorizations’. 2022. Accessed: Nov. 09, 2023. [Online]. Available: https://eprint.iacr.org/2022/1646

[11] ‘ERC-4337: Account Abstraction Using Alt Mempool’, Ethereum Improvement Proposals. Accessed: Nov. 09, 2023. [Online]. Available: https://eips.ethereum.org/EIPS/eip-4337

[12] Y. Hu, M. Liyanage, A. Mansoor, K. Thilakarathna, G. Jourjon, and A. Seneviratne, ‘Blockchain-based Smart Contracts - Applications and Challenges’. arXiv, Jun. 08, 2019. doi: 10.48550/arXiv.1810.04699.

[13] ‘Ethereum Backend Proxy’. Ethersphere, May 25, 2023. Accessed: Oct. 04, 2023. [Online]. Available: https://github.com/ethersphere/ethproxy

[14] orbulo, ‘PaoloRollo/ethlogspy’. Jun. 04, 2023. Accessed: Oct. 04, 2023. [Online]. Available: https://github.com/PaoloRollo/ethlogspy

[15] A. Coglio, ‘Ethereum’s Recursive Length Prefix in ACL2’, *Electron. Proc. Theor. Comput. Sci.*, vol. 327, pp. 108–124, Sep. 2020, doi: 10.4204/EPTCS.327.11.

[16] ‘Deploy Private Amana Network - Qitmeer/Meerlabs Community Documents’. Accessed: Oct. 04, 2023. [Online]. Available: https://meerlabs.github.io/community_docs/deploy_private_amana_network/

[17] ‘Deploy Private Amana Network (AWS) - Qitmeer/Meerlabs Community Documents’. Accessed: Oct. 04, 2023. [Online]. Available: https://meerlabs.github.io/community_docs/AWS_deploy_amana_privnet/