YKYR Smart Contract Wallet: Granular, role-based permission wallet structure

Abstract. The YKYR Smart Contract Wallet is a solution designed to bring unparalleled granularity, security, and flexibility to blockchain interactions. Unlike traditional wallets or existing account abstraction solutions, it introduces a highly granular, role-based permission system that allows users to delegate specific functionalities to their task-specific agents without compromising overall wallet security.

Keywords: Account Abstraction, Proxy Design, TEE

Table of Contents

1. Introduction
2. Background and Motivation
3. Technical Architecture
    - Contract Components
        - YKYRWallet Implementation
        - YKYRWallet Factory
    - Roles and Permissions
        - Owner
        - Proxy Account
        - Relayer Account
    - Data Flow and Interactions
4. Key Features
    - Wallet Initialization
    - Role Management

    - Data Submission Mechanisms
        - Auto Submission
        - Relayer Submission
    - Reward Management
5. Security Considerations
    - Upgradability and Proxy Patterns
    - Signature Verification
    - Replay Protection (Nonce Management)
    - Access Control
6. Use Cases
    - Decentralized Applications (DApps)
    - Decentralized Finance (DeFi)
    - Data Registry Interactions
7. Future Enhancements
    - Potential Integrations
    - Feature Roadmap
8. Conclusion
9. References

1. Introduction

The YKYR Smart Contract Wallet is a decentralized wallet solution crafted to meet the evolving needs of the blockchain community. It offers a secure, role-based system that allows for efficient data submission and interaction with decentralized services. By integrating proxy and relayer accounts, the wallet enhances usability and transaction efficiency, making it an essential tool for users who require advanced functionalities beyond traditional wallet capabilities.

2. Background and Motivation

As blockchain technology matures, the complexity of interactions within decentralized ecosystems has increased. Traditional wallets often lack the flexibility and security features necessary to handle intricate operations such as role delegation, off-chain computations, and secure data submissions.

Challenges in Existing Wallets:

- **Limited Access Control:** Most wallets do not support granular permissions, making it difficult to delegate specific tasks without exposing private keys.
- **Data Submission Cost:** Submitting data directly on-chain can be costly, hindering mass adoption.

YKYR Wallet's Solution:

The YKYR Smart Contract Wallet addresses these challenges by introducing:

- **Role-Based Permissions:** Allows owners to assign proxy and relayer accounts with specific permissions, enhancing security and operational efficiency.
- **Relayer Mechanisms:** Enables off-chain interactions, making gasless transactions possible.
- **Enhanced Security Features:** Implements nonce management and signature verification to protect against replay attacks and unauthorized access.

3. Technical Architecture

Contract Components

YKYR Wallet Implementation

The YKYR Wallet Implementation contract is the core of the wallet's functionality. It defines the logic for role management, data submission, and reward handling. By utilizing an upgradable pattern, the wallet ensures that it can adapt to future requirements without compromising security.

Purpose:

- **Role Management:** Allows the owner to assign or revoke proxy and relayer accounts, controlling who can perform certain actions on behalf of the owner.
- **Tx handling:** Provides functions for interacting with external contracts either directly or via a relayer.
- **Reward Management:** Enables the owner to set an explicit reward address, enabling on-chain KYC if set accordingly.

YKYR Wallet Factory

The YKYR Wallet Factory contract is responsible for deploying new instances of the wallet. It uses the clone (minimal proxy) pattern to create lightweight and efficient wallet instances for users.

Purpose:

- **Wallet Deployment:** Allows for the creation of new wallet instances with minimal gas costs.
- **Initialization Options:** Supports standard initialization and relayer-based initialization, catering to different user needs.

Roles and Permissions

Owner

- **Definition:** The primary controller of the wallet with the highest level of permissions.
- **Responsibilities:**
    - Assigning and revoking proxy and relayer accounts.
    - Managing reward addresses and claiming rewards.
    - Overseeing the overall security and functionality of the wallet.

Proxy Account

- **Definition:** An account authorized by the owner to perform specific actions on their behalf.
- **Responsibilities:**
    - Submitting data directly to data registries.
    - Facilitating operations that require delegated authority.

Relayer Account

- **Definition:** An account that enables off-chain computations and interactions, acting as an intermediary between the owner and the blockchain.
- **Responsibilities:**
    - Executing transactions to perform on-chain.
    - Submitting jobs to TEE pools and handling data submissions via relayer functions.

4. Key Features

Role Management

The wallet allows the owner to manage roles dynamically, enhancing security and adaptability.

- **Assigning Roles:** The owner can assign proxy and relayer accounts, granting them specific permissions.
- **Revoking Roles:** Roles can be revoked at any time, ensuring that access can be restricted if needed.
- **Combined Role Assignment:** The relayer can facilitate the assignment of both proxy and relayer accounts through owner-signed messages, streamlining the process.

Data Submission Mechanisms

The wallet provides efficient methods for submitting data to external services, optimizing for security and cost.

Auto Submission

- **Purpose:** Allows the proxy account to submit data directly to a data registry.
- **Advantages:**
    - **Efficiency:** Reduces the need for the owner to be involved in every transaction.
    - **Security:** Ensures that only authorized accounts can submit data.

Relayer Auto Submission

- **Purpose:** Enables data submission and job execution via the relayer, leveraging off-chain computations.
- **Advantages:**
    - **Cost Reduction:** Minimizes gas fees by handling computations off-chain.
    - **Enhanced Security:** Uses signature verification and nonce management to prevent unauthorized actions.

Reward Management

The wallet includes mechanisms for managing and claiming rewards accumulated through its operations.

- **Setting Reward Address:** The owner can designate an address where rewards will be sent.
- **Claiming Rewards:** Accumulated rewards can be claimed by the owner, providing incentives for the wallet's usage.
- **Transparency:** Events are emitted during reward updates and claims, ensuring transparency in operations.

5. Security Considerations

Upgradability and Proxy Patterns

- **Upgradeable Contracts:** The wallet uses upgradable patterns to ensure that it can be enhanced without losing its state or compromising security. Defining a governance mechanism could later incentivize users to decide if a given version upgrade is needed or not.
- **Immutable Variables:** Critical addresses like the data registry and TEE pool are set as immutable to prevent unauthorized changes.

Tx Execution Protection

- **Signature Verification:** Ensures that only authorized actions are performed by verifying signatures against known accounts.
- **Nonce Usage:** Each transaction includes a nonce, which is incremented after use to ensure uniqueness. **Protection Against Replay Attacks,** by requiring the nonce to match the expected value, the wallet prevents the reuse of signed messages.
- **Role-Based Modifiers:** Functions are restricted using modifiers that check the caller's role (e.g., onlyOwner, onlyProxy, onlyRelayer).
- **Strict Role Enforcement:** Unauthorized attempts to call restricted functions are rejected, maintaining the integrity of the wallet.

6. Use Cases

Decentralized Applications (DApps)

- **Secure Data Management:** DApps can leverage the wallet's ability to securely submit and manage data within decentralized ecosystems.
- **Role Delegation:** Developers can assign proxy accounts to handle routine tasks, improving efficiency.

Decentralized Finance (DeFi)

- **Automated Transactions:** The relayer mechanism allows for automated trading strategies and staking without constant on-chain interaction.

- **Asset Protection:** The wallet's security features safeguard assets against unauthorized access and transactions.

Data Registry Interactions

- **Efficient Data Submission:** Users can submit data to registries with reduced gas costs and improved speed via the relayer.
- **TEE Pool Integration:** By submitting jobs to TEE pools, users can engage in secure, private computations necessary for sensitive applications.

7. Future Enhancements

Potential Integrations

- **Cross-Chain Compatibility:** Exploring interoperability with other blockchain networks to expand the wallet's utility.
- **Oracle Services:** Integrating with oracle providers to access external data, enhancing the wallet's capabilities in DeFi and other applications.

Feature Roadmap

- **Multi-Signature Support:** Implementing multi-signature functionality to require multiple approvals for critical actions, enhancing security.
- **User Interface Development:** Developing intuitive interfaces for interacting with the wallet, making advanced features accessible to non-technical users.
- **Advanced Analytics:** Providing tools for users to analyze wallet activity, helping them make informed decisions.

8. Conclusion

The YKYR Smart Contract Wallet represents a significant advancement in decentralized wallet solutions. By incorporating role-based permissions, relayer mechanisms, and robust security measures, it addresses key challenges faced by users and developers in the blockchain space. The wallet's flexibility and efficiency make it a valuable tool for a wide range of applications, from DApps to DeFi. With planned future enhancements, the YKYR Wallet is poised to continue evolving alongside the rapidly changing landscape of decentralized technology.

9. References

- **OpenZeppelin Documentation:** Guidance on secure smart contract development and upgradable contract patterns.
- **Ethereum Improvement Proposals (EIPs):**
    - **EIP-712:** Standard for typed structured data hashing and signing.
    - **EIP-1167:** Proxy contract for minimal deployment of clones.
- **Solidity Documentation:** Official language documentation providing insights into best practices and language features.
- **Blockchain Security Resources:** Articles and papers on signature verification, nonce management, and smart contract security best practices.