MultiSig Wallet With Onchain Signature Verification

Project Descrtiption
A multi signature wallet smart contract. The contract will help store tokens safely by having multiple people hold ownership of one address or set of funds. Such a wallet will help avoid a single point of failure and split responsibilities of ownership of address and its funds. Multiple user authenticating transactions will also prevent malicious activities.

Problem
As more organizations adopt blockchain especially with the growth in DeFi, the first step is usually the need for a wallet to hold funds. Multi-signature wallets offer a good balance between security and convenience and well suits the structure and needs of an organization. With possible fears of losing keys or theft of a hardware wallet which makes it almost impossible to recover the funds or losing funds when a wallet provider who is the actual owner of your private key is hacked as we have seen in the parity hack of 2017 as well as Crypto exchanges hack of 2019, the need for extra layer of security and recover-ability cannot be overlooked.

Goal
The goal of this project is to apply the learned concepts in advance smart contracts development course including; optimization, security, and efficiency through off-chain computations to develop an improved multisig wallet inculcating best practises and principles.

Other use cases for Multi-Signature wallets

- Voting for decisions: A 4-of-6 multi-signature wallet can let a board vote for decisions. Only when the majority of the board members pass the decision will the funds be allocated.

- Escrowing transactions: A 2-of-3 wallet can be used among the seller, buyer, and a trusted arbiter. If goods are as expected, both the seller and buyer can sign the transaction for payment. If there is a dispute, the trusted arbiter will step in and provide the signature to either party based on the judgment.

- Two-factor authentication as provided by the smart contract wallet implementation.

Disclaimer
This project is a school project and not for production use. While the concepts are true and multisig wallet implementations by ConsenSys and Gnosis were referenced, there is still a lot to do and given the limited time frame for submission, functionalities were prioritized to proof as well as have a useful wallet.

Major Components
In developing the multisig wallet, the following basic components were incorporated:

- Owners: A list of addresses who have the access right
- Approval: Rules to approve submitted transaction (2 of 3, 3 of 4 etc)
- Funding: A means of funding wallet
- Transactions: Submitting and managing transaction.

Code Structure

Events

Name	**Purpose**
Deposit	Logs details of funds to wallet
SubmitTransaction	Logs details of a submitted transaction
ConfirmTransaction	Logs details of approvals
RevokeConfirmation	Logs details of approval removal
ExecuteTransactions	Logs details of executed transaction

State Variables

Name	Data Type	Purpose	Visibility
owners	Array of addresses	List of owners	public
isOwner	mapping	Verify if an address is owner	public
numConfirmationsRequired	uint8	Needed no of approvals	public
Transaction	Struct	Store meta data about transactions	
Transactions	Array of transactions	List of transactions	public
isConfirmed	Mapping of Mapping	Show if an wner has confirmed a transaction	public

Modifiers

Name	**Performs**
onlyOwner	Check if address is an owner
txExists	Checks if transaction exists
notExecuted	Checks if Transaction is yet to be executed
notConfirmed	Check if approvals are complete

Functions

Name	**Purpose**	**Visibility**
_addTransaction	Add a transaction to transaction pool	private
submitTransaction	Relies on _addTransaction to submit a transaction	Public onlyOwner
confirmTransaction	Confirm a transaction	Public onlyOwner
revokeConfirmation	Revoke a confirmation	Public onlyOwner
executeTransaction	Execute a Transaction	Public onlyOwner
verifySignature	Verify a signature and add message to the transaction pool	Public onlyOwner
getTransaction	Retrieve details of a transaction	Public
receive	Receive fund	public

Project Structure
Standard truffle project folders with addition of coverage report folder as well as a scripts folder which contains a script to generate an off chain signature.

Development Tools

- Truffle
- Javascript
- Solidity

Smart Contract Vulnerabilities handled and other improvements incorporated

- Protected functions by specifying roles that should be allowed to invoke the function and current state of data/transactions.
- Appropriate data types to optimize storage and prevent exploits.
- Arranging variable and data declarations to ensure optimal packing in storage
- Transaction ordering to prevent race conditions
- Mitigated replay attacks through checked nonces and contract address inclusion in hashed message.

How to run
Running the project follows similarb steps with every other truffle projects.

Steps:

1. npm install: to install dependencies
2. truffle migrate –reset: to deploy to local blockchain
3. truffle test: to run tests
4. truffle run coverage: to see test coverage report
5. In order to test the verifySignature function, you will have to use remix and connect to meta-mask and be connected to the local running blockchain to have access to private keys.
   Get private key of an owner address and use the **signOffchainTransaction** script in the scripts folder, provide the required inputs and generate signature components. Pass the generated outputs to verifySignature function and a new transaction should be submitted if valid.

To Do

- Fix automated test for verifySignature function.
- Modify verifySignature function to use inline assembly and save gas cost.

References

- Ethereum Cookbook By Manoj P R
- Ethereum Smart Contract Development By Mayukh Mukhopadhyay
- Class demo & lab notes: https://github.com/GeorgeBrownCollege-Toronto/Advanced-Smart-Contracts
- https://github.com/ConsenSysMesh/MultiSigWallet
- https://multis.co/post/multisignature-wallets-the-gold-standard-for-companies
- https://github.com/gnosis/MultiSigWallet
- https://smartcontractprogrammer.com/courses/