BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION German Informatics

DIPLOMA THESIS

2FA Solidity Multisig Wallet

Supervisor
Grad, MIHAI-FLORIN-GABRIEL CRĂCIUN

Author
EȘAN TUDOR-CONSTANTIN

2024

ABSTRACT

In the context of cryptocurrency security, multisig (multiple signature) wallets have gained popularity due to their ability to provide a higher level of security compared to traditional wallets. These wallets increase the security of digital funds by requiring transaction approval from numerous sources. The introduction of the Atlas feature, a two-step authentication (2FA) method for transaction approval, represents a new development in multisig wallet technology. This adds an extra degree of protection by fusing the benefits of multisig management with the efficiency and user-friendliness of 2FA solutions commonly found in web2 apps. Using Atlas in multisig wallets improves security while simplifying the approval process and making transactions easier to access and handle. With its potential to revolutionize the authorization and security of cryptocurrency transactions, this approach offers a strong means to manage risk in the rapidly developing cryptocurrencies world.

Contents

Chapter 1

Introduction

1.1 Motivation

Cryptocurrencies, digital assets for global money transfers, are reshaping how we think about finance. They use decentralized networks, allowing for faster, cheaper transactions without banks. Key benefits include almost instant payments, lower fees by eliminating middlemen, and the ability to transact worldwide.

The cryptocurrency journey began in the 1980s, revolutionizing in 2009 with Bitcoin's introduction by Satoshi Nakamoto. Bitcoin, leveraging blockchain technology, has become the best investment over the past decade, leading to widespread adoption and recognition, including the historic approval of the first Bitcoin ETF in January 2024. This not only showcases Bitcoin's status as a premier asset but also signals the mainstream financial world's embrace of cryptocurrencies.

Ethereum, introduced after Bitcoin, revolutionizes with smart contracts—automatic agreements executing conditions without human intervention. This technology enables decentralized applications (DApps) and decentralized finance (DeFi), transforming everything from voting systems to direct artist-to-fan sales without intermediaries.

Running a business without traditional banks is not without its difficulties, though. The secret digital signatures known as private keys are what make cryptocurrencies secure. The majority of users own non-custodial wallets, which give them complete control over their money and exclusive access to the private key. Although there are several advantages, there are also disadvantages. A person who loses their private key will no longer be able to access their entire fund, and since cryptocurrencies are decentralized, nobody can assist you. The fact that you won't be the only person with access to the money if your private key is taken and that you can't take back the money once it leaves your wallet is another issue, as discussed in the article by NEFTURE SECURITY [NEF24]

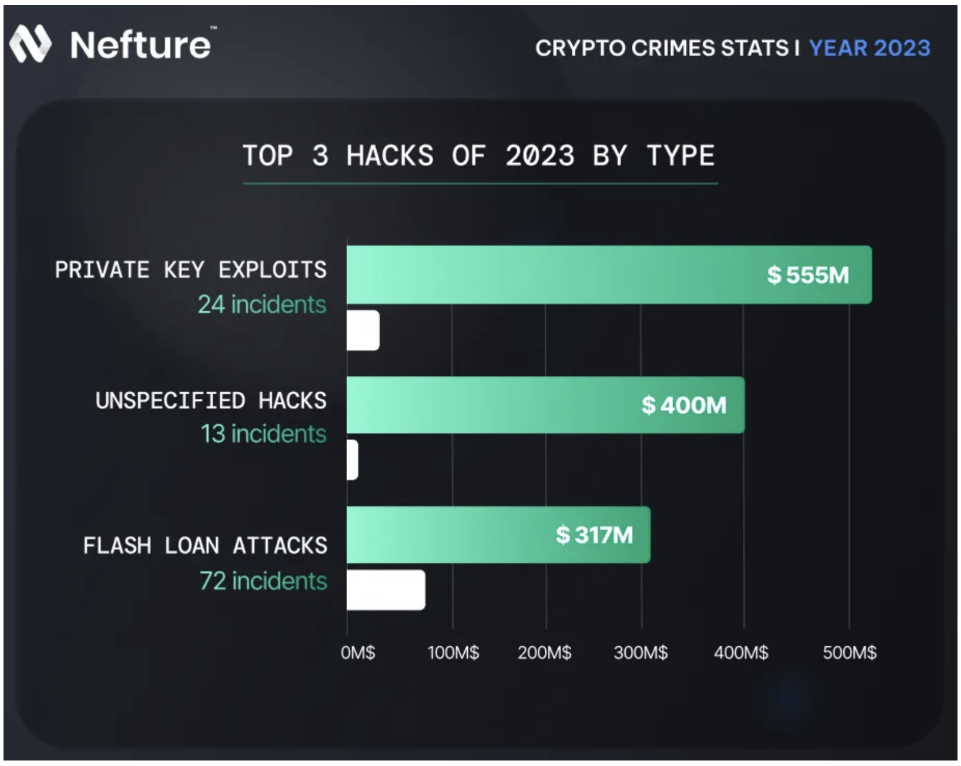

Figure 1.1: Total stolen funds value

A notable aspect that needs to be mentioned is that according to Nefture "Only 16.6% of the contracts analyzed were managed by multisig wallets"[NEF24]. Using a multisig wallet can drastically reduce the probabilty of getting hacked especially if it was set up with multiple ownsers. This method lowers the possibility of unauthorized access to the system by requiring the consent of many parties prior to transactions being completed, adding an extra layer of security.

1.2 Purpose

The aim of this thesis is to develop a multisig wallet called "Vault" compatible with any EVM (Ethereum Virtual Machine) chain. Vault will provide a secure and efficient solution for managing digital assets across a wide range of blockchain platforms.

Vault will be a unified platform for managing digital assets across multiple EVM blockchains such as Ethereum, Base, Polygon etc. Users will be able to store, transfer and receive cryptocurrencies and NFTs easily and securely using a single interface.

Vault is a simple to use multisig wallet that offers increased flexibility and security for managing digital assets. Unlike traditional wallets, Vault does not rely on a single private key, but uses a smart contract to distribute responsibility between multiple people or devices. This approach allows for personalized wallet configuration, perfectly adapting to individual needs and providing granular control over access to funds. Using smart contracts for this also allows for infinite configuration in the future. For features like: Key Recovery, Accounts Whitelisting (trusted accounts the wallet is allowed to interact with), Spending Limits, Subscriptions etc.

Like any other multisig wallet the number of owners, and the approval threshold (the minimum number of signatures (approvals) execute a transaction) can be specified. This enables Vault to be used in a lot of different use cases:

- Shared Accounts: To enable cooperative fund management, several owners can be set up with a low approval requirement (for example, 1 of 2).
- Enhanced Security: In order to guarantee access to funds even in the event of a lost private key, users can designate multiple private keys as owners. To improve security, a higher acceptance threshold (such as two out of three) might be specified.
- Corporate Governance: The company's money can be transparently and democratically controlled by a board of directors acting as wallet owners, with a majority (i.e., 50% + 1) vote required to approve transactions.

1.3 Related work

Multi-sig wallets offer a more secure solution for managing digital assets by sharing control between multiple people or devices. This section reviews relevant work related to multisig wallets, blockchain security protocols and multi-channel interoperability, contextualizing Vault in the existing landscape.

Currenlty some of the most used multisig wallets are:

- Gnosis Safe: Since its 2018 launch, users have come to trust it because of its strong security and adaptable features. DApp integration: Allows for use in a variety of DeFi, NFT, and DAO scenarios by integrating with different DApp platforms. Smart contracts that have been audited: The wallet's smart contract code has been examined by recognized professionals, which lowers the possibility of security flaws. Decentralized governance: SafeDAO oversees the wallet's development, guaranteeing openness and participation from the community.
- Rabby Wallet: "A Web3 wallet called Rabby Wallet provides a seamless multichain experience by automatically navigating to the relevant chain based on the Web3 dApp that you have visited. You may verify faults and hazards prior to signing transactions with our security rule engine. When you sign a transaction, Rabby Wallet displays the estimated balance change to you."[rab]

- Argent X Wallet: ”Argent is the first non-custodial wallet with no seed phrase and no complexity. With your Argent Vault, enjoy peace of mind through locking and unlocking your account, auto blocking transactions, and setting trusted contacts:”[arg]

When compared to the other wallets, Vault’s interface is far more straightforward and user-friendly. However, the Atlas 2FA feature is the best feature that no one else has. To put it simply, Vault allows you to set up a two-factor authentication code with Google authenticator. This code is required in addition to standard multisig security in order for the transaction to be completed. The best part is that you can configure your multisig to function as a wallet with one owner and one approval signature, but you can also turn on the Atlas for an additional security layer.

Chapter 2

Blockchain

2.1 Generalities

A blockchain is a growing collection of items known as blocks that is stored in a decentralized, transparent, and secure ledger. Each block is linked to the previous one using hashes to form an unbreakable chain. Blockchains come are useful in a lot of scenarios where having a centralized middleman is not required. Currently, peer-to-peer currency represent the main use case for blockchain technology. cryptocurrencies that keep track of all user balances and transactions without the assistance of a single third. Bitcoin was the first blockchain of this kind as was defined as: "A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution"[Nak08]. The main characteristics of Bitcoin are:

- Decentralisation: No sigle entity controlls the blockchain. It is managed by a distributed network of nodes.
- Imutability: Once a transaction is recorded on the blockchain it is almoast impossible to change it in the future
- Transparency: Everyone can view every transaction on the blockchain and everyones balance since the begining of the blockchain
- Security: Data on the blockchain is protected from modification and hacking by strong encryption.

With only 21 million bitcoins ever created, it is currently considered digital gold. Bitcoin was a groundbreaking idea. However, it is not without its shortcomings. Among them include the fact that Visa offers 24000 TPS whereas Bitcoin only allows 7 TPS (transactions per second). Another disadvantage is that fees are paid in Bitcoin. This meant that transactions were quite inexpensive when it initially launched, but since then, the price has increased from roughly zero to seventy thousand dollars, and fees have also increased. At the moment, a transfer costs about $6, but this is becoming worse every day. Because of this limitations since then over 1000 blockchains appeared according to Watcher Guru [wat]

Ethereum is the 2nd blockchain by market capitalization after bitcoin. After overcoming the early constraints of Bitcoin, the introduction of Ethereum signaled a turning point in the development of blockchain technology. Co-founder of Ethereum, Vitalik Buterin, pointed out that blockchain technology has applications beyond just transferring money. His goal was to build a programmable computer running code on top of Ethereum, a flexible framework for decentralised applications (DApps).

Buterin introduced the idea of a programmable blockchain with its own programming language, Solidity, for creating smart contracts in the Ethereum whitepaper, which was published in 2013. The description of Ethereum is "A Next-Generation Smart Contract and Decentralized Application Platform." [But14]

The possibilities for blockchain are endless thanks to the capacity to write self-executing smart contracts. At its core, Ethereum is a worldwide network of nodes that can execute smart contract code thanks to the Ethereum Virtual Machine (EVM). Transaction fees or "gas" are paid for with ether money (ETH), which is also used to compensate network miners. Ethereum is now the foundation of a flourishing DApp ecosystem that is promoting innovation across many industries. The idea of Web3, which envisions a more user-based, decentralized internet, has been strengthened by it.

2.2 How do cryptocurrencies work?

Blockchain uses public and private key cryptography to ensure the security and integrity of transactions. This cryptography system is essential to the operation of blockchain technologies, including cryptocurrencies. Here's how it works:

1. Key generation: A pair of cryptographic keys—a private key and a public key—are generated for every blockchain user - tipically by a crypto wallet. While the public key is open for distribution, the private key is confidential and needs to be kept secure by the user. Although there is a mathematical relationship between these keys, the private key cannot be obtained from the public key. With the public key a unique address is generated.

2. Transaction creation: When a user wants to initiate a transaction, for example, send tokens to another account, they create a transaction message. This message includes information such as the recipient's address, the amount transferred and other optional data.

3. Signing the transaction: With their private key, the user signs the transaction. Each transaction generates a unique digital signature using the private key, which acts as mathematical evidence that the person whose private key is linked to the account from which the money is being transferred approved the transaction.

4. Transaction Verification: Using the sender's public key, the network's nodes—or miners, in the case of Bitcoin—verify the transaction's digital signature after it has been published on the blockchain. The transaction is regarded as authentic and can be completed if the signature is legitimate.

5. At last, the transaction is added to the blockchain by being included in a new block that is subsequently appended to the current chain of blocks upon validation—thus the term "blockchain." The transaction becomes permanent and irrevocable once it is recorded to the blockchain, offering a transparent and safe record of the money transfer.

Blockchain utilizes cryptography in these ways to offer a decentralized, transparent, and extremely safe payment system without the need for a dependable central authority like a bank or other financial middleman.

2.3 Smart Contracts

For a person that has not much experience with blockchain development, smart contracts are functions that are deployed on the blockchain and can be called by everyone. After the deployment the smart contract code cannot be altered. Investopidia defines smart contracts as: "a self-executing program that automates the actions required in an agreement or contract. Once completed, the transactions are trackable and irreversible. Smart contracts permit trusted transactions and agreements to be carried out among disparate, anonymous parties without the need for a central authority, legal system, or external enforcement mechanism"[sma]. Some applications of smart contracts are the following:

Decentralized Finance (DeFi):

- Liquid Staking: Users can stake their ETH on websites like as Lido or Rocket Pool, and in exchange, they will obtain liquid tokens equivalent to their ETH worth. Then, by using these liquid tokens in different DeFi protocols, more returns can be produced.

- Descentralized exchanges: Users can exchange various crypto-assets directly with one another without the necessity of centralized middlemen thanks to decentralized exchanges (DEX) like Uniswap and SushiSwap

- Lending and borrowing: Users can deposit or lend cryptocurrency assets to earn interest through platforms like Aave or Compound. All transactions are openly managed by smart contracts.

NFT Markets: Using smart contracts, marketplaces such as OpenSea or Rarible enable the buying, selling, and trade of unique tokens (NFTs), which stand in for digital art pieces, collectibles, or game assets.

Decentralized autonomous organizations (DAO), such as MakerDAO, use smart contracts to enable token holders to speak up for themselves and cast votes on decisions that will determine the protocol's future development.

Blockchain gaming: Smart contracts facilitate the management of game economies, include NFT components into the game, and guarantee a fair and transparent game for all players.

2.4 Crypto Wallets

Cryptocurrency wallets are digital tools that allow users to store and manage their cryptographic keys used to transact with cryptocurrencies such as Bitcoin, Ethereum, and others. These wallets can exist in various forms, from software applications to hardware devices, and are essential for using cryptocurrencies in a safe and efficient way. Shortly put, a crypto wallet hold one or more private key for you to interact with the blockchain.

Wallets for cryptocurrencies serve a number of crucial purposes:

- Security: Offers a safe space to save private keys, which are necessary to when using cryptocurrencies, because whoever access the private key has full access to the funds.

- Interacting with the blockchain: Make it possible to send and receive money using cryptocurrencies, enabling transfers, or DApps interactions.

- Interoperability: Enables users to engage with many blockchains and take part in multiple ecosystems such as Bitcoins, Ethereum etc.

- Control: Gives users complete control over their digital assets without the need for intermediaries such as banks.

Cryptocurrency wallets fall into several categories, depending on the nature and level of security offered:

1. Software wallets: These are computer or mobile applications that can be downloaded. Although they are practical, their degree of security is dependent on

how secure the host device is. For instance, if your computer is compromised by malware or viruses and you have an extension wallet, the private key could be taken.

2. Hardware wallets: These are physical devices that store and generate private keys offline, providing increased security. The private key is also stored on a hardware device and it never leaves it. They are considered some of the safest options for long-term cryptocurrency storage.

3. Paper wallets: Involves printing private and public keys on paper, which are then stored in a secure location. They eliminate the risks associated with cyber attacks, but are susceptible to physical risks such as damage or loss. Almost every time a hardware wallet will be better than this.

To enable access to and trade of cryptocurrencies, private key wallets employ pairs of private and public keys. They offer a high level of security as long as the private key is safeguarded and kept secret, and they are simple to use and the foundation of the majority of bitcoin wallets currently in use.

However, smart contract wallets handle transactions using smart contracts on Ethereum and other platforms. Advanced capabilities that these wallets can offer include approving transactions from several parties at once or incorporating intricate trade rules straight into the contract logic. Smart contract wallets are perfect for sophisticated usage, such as automatically managed money or wallets that need several approvals to conduct a transaction, because of their considerable customization and automation options. Even if smart contracts wallets are much more sophisticated, to interact with them you still need to use a wallet with a private key, however, they can add features to increase security.

2.5 Multisignature Wallets

Multisig wallets, sometimes referred to as multi-signature wallets, are an important development in the security of cryptocurrencies. They improve the security of monies that are kept in storage by requiring the consent of several parties before executing a transaction.

A smart contract that specifies how many of the preset holders must sign a transaction before it is performed is the foundation of a multisig wallet. A wallet can be set up, for instance, so that two of the three holders must sign it. This system strikes a balance between security and accessibility by preventing illegal acts and preventing the loss of access to funds in the event that a single key is compromised.

Comparing multisig wallets to conventional single-signature wallets reveals numerous noteworthy benefits. The most important thing is more security. The possibility of theft or fraudulent transactions is significantly decreased by needing numerous permissions. Multisig wallets are perfect for groups or organizations that manage shared assets since they also make it easier for members to collaborate and manage funds together.

Chapter 3

Technologies

3.1 Solidity

Solidity is a programming language that was created specifically for Ethereum smart contract creation [sdo] .The primary features are:

- OOP: Solidity is an object-oriented programming language, which makes code organization and reuse simple. It is built on the idea of classes and objects.
- JavaScript-Like: Because of its core structure, which is similar to that of JavaScript, web professionals may easily learn Solidity. Its quirks have been tailored especially for blockchain, nevertheless.
- Solidity utilises static typing, which necessitates the specification of the data type of variables in advance. By doing this, compile-time errors are caught and contracts are enforced..

However given that smart contracts are deployed once and cannot be altered after you need to be very carefull about potential security risks that can lead to stolen funds. Some of the most common vulnerabilities are: Reentrancy Attacks, Integer Overflow/Underflow, Uninitialized Storage Pointers, Denial of Service (DoS) Attacks [sola]

3.2 Hardhat

Hardhat is an open-source testing and development framework for Ethereum. The open-source framework for Ethereum development and testing [har]. It improves the process for creating and testing smart contracts. Hardhat is flexible; it supports numerous Solidity compilers and allows for different network configurations. Additionally, it may be expanded upon by combining it with additional blockchain ecosystem tools and frameworks. The comprehensive range of testing features contributes to the security and reliability of smart contracts. Hardhat strongly speeds up development, enhances the quality of the code, and lowers the possibility of mistakes in contracts that are well-written.

3.3 Nextjs

Next.js is an open-source React framework that extends the core React capabilities to create high-performance and SEO-friendly web applications. NextJs allows components to be rendered on the server, not only does it help with speed because the server can prerender the response but it can also help with webcrawlers to scan your site. The best features of NextJs include:

- Image optimizations: "Automatically serve correctly sized images for each device, using modern image formats like WebP and AVIF." [nexa]
- Dynamic HTML Steaming
- Client And Server Components
- Server Actions: "Server Actions are asynchronous functions that are executed on the server. They can be used in Server and Client Components to handle form submissions and data mutations in Next.js applications." [nexb]
- Route Handlers: "allow you to create custom request handlers for a given route using the Web Request and Response APIs." [nexc]

3.4 Typescript

"TypeScript adds additional syntax to JavaScript to support a tighter integration with your editor. Catch errors early in your editor"[typ]. Static types contribute to the reliability and maintainability of your code by reducing compilation errors. Classes make code more readable and scalable by enabling it to be divided into reusable modules. Compiling TypeScript into plain JavaScript makes it interoperable with every runtime and browser. Using TypeScript has several advantages:

- Boost code quality: Static types make code more robust by assisting in the detection and prevention of compilation issues.
- Boosts scalability: Code may be arranged into reusable modules by using classes and modules, which makes it simpler to create bigger projects.

- Boost teamwork: Integrated documentation and static types can help enhance teamwork and communication.

3.5 Tailwind

Tailwind is defined as "a utility-first CSS framework packed with classes like flex, pt-4, text-center and rotate-90 that can be composed to build any design, directly in your markup" [tai].

Some of the Tailwind CSS's primary features are:

- Utility Classes: An extensive library of pre-made classes that style HTML components without requiring the creation of unique CSS code.
- Components that can be customized: Enables the development of reusable parts with unique styles.
- Expandable: Plugins can be added to expand the capabilities.

The Benefits of tailwind is that it drastically accelerates development, less CSS code writing is needed. There is no need in thinking what to name your css classes because they are pre defined. It minimizes the CSS file size, only the css needed is kept, the rest is "purged"

3.6 RainbowKit

RainbowKit is an open-source SDK that makes it it easier to incorporate Web3 wallets into your web apps. It gives consumers a range of alternatives by supporting multiple blockchains, including nearly every EVM compatible chain. RainbowKit's user-friendly interface lowers friction while facilitating a simple and seamless connection experience. Because of its customization capabilities, you can incorporate Web3 features unique to your project and alter the wallet's appearance. Rainbowkit also allows to easily change your chain whenever you like [rai]

3.7 Infura

Infura is a platform that gives you access to blockchain nodes, drastically simplifying the process of connecting and interacting with blockchains such as Ethereum, IPFS, and Polygon, etc. By using Infura, you can save time and money by avoiding the hassle of configuring your own nodes. With the help of this platform's high scalability, security features, and quick access to blockchain data, you may create decentralized apps (dApps), incorporate blockchain technology into already-existing apps, and test concepts quickly without needing to design complicated infrastructure. [inf]

3.8 Etherscan

The Etherscan API allows developers to access and query data from the Ethereum blockchain. It offers a wide range of functions that allow:

- Get details about particular blocks, transactions, and accounts: You may find out information about balances, contract codes, and transaction histories for particular blocks, transactions, and accounts.
- Sending Transactions: You can call smart contract operations and transmit ETH transfers to the Ethereum network.
- Subscribe to events: You can get alerts in real time when new blocks or ETH transfers occur on the blockchain.
- Historical data query: The Ethereum blockchain provides historical information on blocks, transactions, and account statuses.[eth]

3.9 Viem

Viem is a library that makes it easier and more natural for developers to work with the Ethereum network. Viem offers an abstraction over the implementation of the rpc methods used by EVM.[vie] Important aspects of Viem:

- Higher-level abstractions: Viem makes it simpler to work with notions like accounts, transactions, and smart contracts in your code by providing higher-level abstractions for them.
- Defined Data Types: Viem increases the safety and readability of your code by predefining the data types that ethereum will use.
- Multi-Chain Support: Polygon and Avalanche are two of the many Ethereum-compatible blockchains that Viem supports.
- Simple integration: Viem interfaces with several well-known JavaScript frameworks and libraries with ease.

3.10 Wagmi

Wagmi is a library that greatly simplifies web3 development when utilizing React. To make communicating with the blockchain easier, Wagmi provides over 20 react hooks. These hooks can be used to get transactions, listen to events, or even communicate with other addresses or smart contracts. Additionally, Wagmi is the authorized connector for EIP-6963, walletconnect, and metamask. Tanstack useQuery enables Wagmi to assist with caching the get requests and avoiding deduplication. [wag]

3.11 useQuery

useQuery is a library for the react framework that helps you with fetching data from external servers and doing mutations. It helps you with a lot of common task when dealing with this kind of use cases. This are some of its features

- Prevents deduplication thanks to query keys
- Data Caching
- Error Handling
- Loading and Refetching handling
- Revalidating data after mutations
- Time based revalidation

Chapter 4

″Vault″ Multisig Implementation

4.1 Multisig Smart Contract

In order to set up the multisig wallet, we have to create a smart contract that runs on the blockchain and complies into bytecode. Let's examine each of the key components of a multisig smart contract:

1. Storage Variables

The following are some important storage variables and their purposes in my specific contracts:

- Owners: During deployment, an immutable list of addresses is initialized that lists the people who are allowed to communicate with the multisig instance.
- numConfirmationsRequired: A number that specifies how many approvals transaction need to have until it can be executed.
- Transactions: A list of structures containing all relevant transaction data: value (in Ether), data, the destination address, and the current number of approvals.

2. View Functions

An essential component of Solidity are the view functions, which let you query data from smart contracts without altering their state.

Key Characteristics of view functions are:

- Do not alter state: View functions, in contrast to regular functions, are unable to alter contract state variables like storage or account balances.
- They don't require a transaction to be called directly by other smart contracts or online interfaces.
- Unlike functions that change states, calling them does not cost any gas

Some examples of the view functions used in the ″Vault″ Multisig Contract are: getOwners, getTransactions, getApprovedOwners (returns all the owners who approved a specific transaction)

3. Regular Functions

Functions make it possible to manipulate data and carry out actions that change the contract's state, they are crucial to smart contracts. The key functions of the multisig contract are:

- submitTransaction(address _to, uint256 _value, bytes memory _data): This is the first transaction you need to call in order to propose a transaction to be approved by the owners.
- confirmTransaction(uint256 _txIndex): An owner-specific call will be made to this function to authorize a transaction. Take note that nothing else is required—just the transaction ID. This is because you can determine which address started the transaction by accessing msg.sender.
- revokeConfirmation: This feature will revoke an approval.
- executeTransaction(uint256 _txIndex): This function will finally carry out the transaction once the necessary number of confirmations has been received.

3. Solidity Events

Solidity events are an important feature in the Solidity programming. They allow smart contracts to communicate with outside applications via ″logs″ that are stored in the blockchain [solb]. They are very usefull because you can query them to see what transactions happend in the past but also listen to them in real time to update the interface in real time. Some logs this smart contract uses are: Deposit, SubmitTransaction, ConfirmTransaction, ExecuteTransaction. This events make it really usefull to query when any sort of action happend in the multisig contract

4.2 Atlas - 2FA Feature

Atlas is a 6 digits one time password that you can set up for your multisig to provide additional security. It is a opt-in feature and you need to enable it yourself if you want to use it.

How did I come up with this idea?

Another blockchain called MultiversX, formerly known as Elrond, served as the model for Atlas. ″Guardian″ is the name of a comparable feature that MultiversX implemented. Any account can be configured with guardians to add further security. ″The Great Heist″ was arranged as an event to demonstrate the extra degree of security. The main goal of the event was to build up a wallet holding more than \$20,000. After the guardian was in place, the MultiversX team tweeted the account's mnemonic. Everyone was informed that the money will be retained by the first person to pass the guardian. But none was able to overcome it. This was an excellent example of the guardian OTP's security.

How do One Time Passwords work (OTP)?

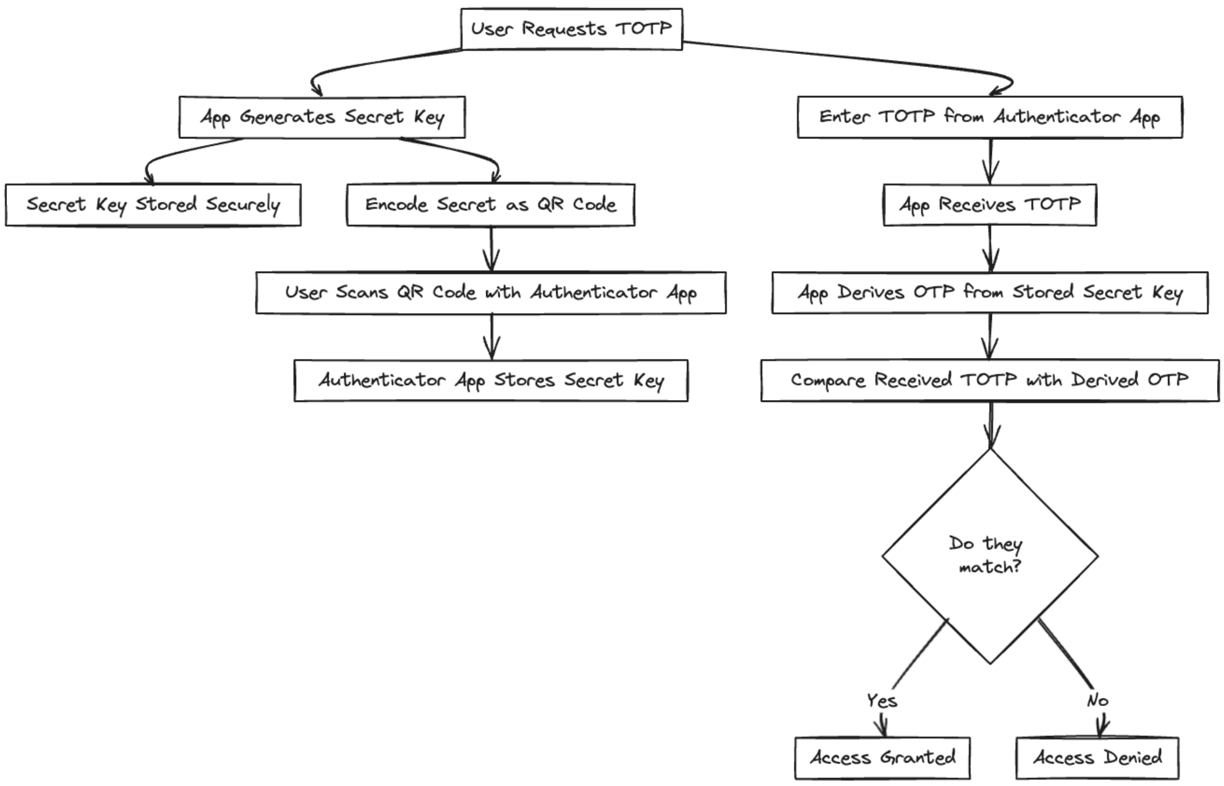
User Requests TOTP
App Generates Secret Key
Enter TOTP from Authenticator App
Secret Key Stored Securely
Encode Secret as QR Code
App Receives TOTP
User Scans QR Code with Authenticator App
App Derives OTP from Stored Secret Key
Authenticator App Stores Secret Key
Compare Received TOTP with Derived OTP
Do they match?
Yes
No
Access Granted
Access Denied

Figure 4.1: Generating and Validating OTPs

Server side A secret key, which is a string of alphanumeric characters, is created by the server and is specifically linked to the user's account. It generates a QR code with this secret key that mobile authenticator apps can use to set up the OTP. Additionally, the server develops a function that will verify any OTP codes that are generated for that specific moment in time with the proper code that is received from the user.

Client side Using an authenticator app such as Google Authenticator, the user scans the given QR code. This app creates unique OTP codes by storing the secret key and using it in conjunction with the current time. Later, when they try to log into the online platform, the user enters the OTP code that the software created.

OTP Validation Process The user enters the OTP code from the authenticator app in addition to their standard password. The server then verifies that the password matches the one linked to the account and determines whether the provided OTP code is valid for the current time window. The user is granted access by the server if both the password and the OTP are verified. Successful verification depends on the server and the device issuing the OTP being in sync with one another in terms of time. On the server, the secret key needs to be safely kept and protected from any potential unwanted access. To ensure reliability, standardized algorithms (TOTP or HOTP) should be used for OTP generation.

How to implement Web3 OTP's

To use Atlas, you must first validate, first configure an OTP using an authenticator such as google authenticator. After the authenticator has been activated and validated, the server will remember the private key that corresponds to the multisig wallet address. To activate the otp on chain, the server will send you the etherum address corresponding to the private key with which the otp is generated to the client, and the client will "propose" an Atlas activation by calling the endpoint: proposeAtlasActivation(address _atlasAddress). For any change of the atlas a timeout period is implemented, we will talk about it below. After the timeout has passed, the activateAtlas() endpoint can be called to activate Atlas, and consequently no transaction from now on will be able to be approved without validating the OTP code. If somehow the user loses access to the used authenticator, he could propose a deactivation of atlas, which will also be subject to the same timeout period, until it can be permanently deactivated.

How does Atlas help?

In the unlikely event that your private key is obtained by an adversary, Atlas comes in quite handy. In this instance, if Atlas was enabled, he would not have been able to take money out of your account without also verifying Atlas. In this instance, the thief must first disable the atlas, which is governed by the previously stated timeout duration. During this period, the account holder will be able to move all of the money to a different address and prevent any money from being stolen because they will have access to both the private key and the OTP.

First Attempt: Confirming Atlas on the server

My first attempt was to set up a POST request on my server that would validate the OTP for a certain multisig wallet. If the otp was correct the server would sign a transaction with the private key that generated this otp and send this transaction to the multisig contract. However this has a major setback. For the account generated with the OTP private key to interact with the multisig contract this would need to have Ether in it to cover the transaction fees. I decided that this approch is not optimal because it is very user unfriendly. The user should not know that atlas is actually another account that needs to be constantly be topped up to work.

Second Attempt: Sending a signed message to the client

After thinking a bit how to not make the private key on the server cover the transaction fees on the server I finally came up with a solution that I think its much better.

The solution was changing the multisig contract. At first the confirmAtlas endpoint had just one parameter, the id of the transaction the user would want Atlas to confirm it. Moreover the confirmAtlas could only be called by the address generated from the private key of the otp. After the modification the confirmAtlas still had the id of the transaction, but it also had an additional parameter named ”signature”. The solution I found was to sign the id of the transaction the user would want to confirm on the server and than return it to the client. The client would call the confirmAtlas endpoint with the index, and the signature and the smart contract would validate that this signed message was signed by the private key of the atlas account. The code looks like this:

```
function confirmAtlas(
 uint256 _txIndex,
 bytes memory signature
) public {
  require(atlasAddress != address(0), "Atlas not activated");
  require(_txIndex < transactions.length, "tx does not exist");
  Transaction storage transaction = transactions[_txIndex];
  require(!transaction.atlasConfirmed, "Atlas already confirmed");

  bytes32 messageHash = keccak256(abi.encodePacked(_txIndex));
  bytes32 ethSignedMessageHash = keccak256(
  abi.encodePacked("\x19Ethereum Signed Message:\n32", messageHash));

  (address recoveredAddress) = recoverSigner(ethSignedMessageHash, sign
  require(recoveredAddress == atlasAddress, "Invalid signature");

  transaction.atlasConfirmed = true;
}
```

This is a major improvement because now the atlas account does not need to have any ether in it so that it can approve a transaction

Common Questions

1. What occurs if someone steals the Atlas Private key? Should that prove to be true, there wouldn’t be a significant security issue. The Atlas private key is limited to approving transactions; it cannot initiate new transactions or communicate in any manner with the multisig contract.

2. Is the Atlas feature centralized? Yes, all the private keys are stored in a postgres database, nevertheless, this is not a security risk, as was mentioned in response

to the initial question. Furthermore, you have the option to turn off the Atlas feature and continue using your account normally in the unlikely case that this third-party service stops functioning.

4.3 Backend

Because most of the logic is handled by the multig smart contract I did not have to implement a lot of functionalities on the backend. Smart contracts can automate a large portion of business logic, particularly that which involves safe and verifiable transactions or interactions. They use the blockchain directly to carry out actions in an immutable and transparent manner. Many of the procedures that were needed to communicate constantly with a backend server are now decentralized thanks to smart contracts. This lessens potential points of failure and attacks that could interfere with the application's ability to function. In an application for a multisig account the smart contract handles 90% of the logic. The only need for the backend was to store the private keys for the Atlas feature and to cache some information to not make a lot of request to the Infura API.

Why NextJS? For my project's backend, I decided to use Next.js for a few reasons. Firstly, Next.js provides amazing ability when it comes to content rendering. By sending pages fast and effectively, entities may maximize speed and user experience with Server-Side Rendering (SSR) and Static Generation (SSG) choices. For my applications, where quick interactions and real-time data display are crucial this is necessary. I can manage backend logic using serverless functions instead of constantly operating a typical server thanks to Next.js. This implies that they are able to manage dynamic requests and interactions with the database or other APIs in a manner that is more economical and efficient. Scalability becomes easier because functions are only called upon when necessary, preventing resource waste and the related expenses of a server operating constantly.

For the Atlas feature instead of using http routes I mostly used "Server Actions". Server actions makes it very easy to write functions that are executed in the server in the same project where your frontend is located. This is an example of a server function and how it looks like:

```
"use server"
export const getConfirmAtlasSignature = async (
  otp: string,
  txIndex: number,
  multisigAddress: string,
  chain: number
```

```
) => {
  const isValid = checkOTP(otp, multisigAddress);
  if (!isValid) {
   throw new Error("Invalid OTP");
  }
  const wallet = (
  await db
  .select()
  .from(WalletsTable)
  .where(sql`${WalletsTable.walletAddress} = ${multisigAddress}`)
  )[0];
  if (!wallet.secret) {
   throw new Error("Secret not found");
  }
  if (!wallet.atlasAddress) {
   throw new Error("Atlas address not found");
  }
  const hexSecret = Buffer.from(
  base32Decode(wallet.secret, "RFC4648")
  ).toString("hex");
  const provider = new ethers.InfuraProvider(
  chain,
  "b3615957c6b0427eb2fac15afb451acb"
  );
  const signer = new ethers.Wallet(hexSecret, provider);
  const messageHash = ethers.solidityPackedKeccak256(["uint256"], [txIn
  const messageBytes = ethers.getBytes(messageHash);
  const signature = await signer.signMessage(messageBytes);
  return signature;
};
```

As it can be seen the ″use server″ expression indicates that the functions declared in the file will only be used on the server, as can be observed. This causes the function to retrieve data or perform updates on it from sources such as the Postgres database. This function can be used directly in a button onclick handler and nextjs will handle the comunication.

The provided function is also how the Atlas confirmation is implemented on the backend. This is how it works

1. Verify OTP: The function begins by determining whether the user-provided

one-time password (OTP) is valid. In order to verify that the individual attempting to complete a sensitive activity or execute a transaction is, in fact, the authorized user, this check is crucial for the 2FA functionality. An error is raised right away if the OTP is invalid.

2. Retrieve Wallet Information: The function then goes on to obtain wallet data from a database. It precisely looks for a wallet that corresponds to the given multisigAddress. This address usually refers to a multi-signature wallet, which increases security by requiring multiple keys to allow transactions. Errors are raised appropriately if the wallet lacks linked secret keys or an Atlas address.

3. Decode Secret Key: The base32-formatted secret linked to the wallet is subsequently converted to hexadecimal. The secret key must be in a certain format for the subsequent cryptographic procedures, which makes this conversion required.

4. Create Blockchain Provider and Signer: This function uses Infura and Ethers.js to create a blockchain provider. Using Infura, you can interact with the Ethereum blockchain without having to manage your own node by using their scalable blockchain node network. To generate a new Ethereum wallet instance (signer) that may sign messages or transactions, utilize the decrypted secret key.

5. Get the message ready to sign: The transaction index (txIndex) is hashed using a technique that works with Solidity, the Ethereum smart contract language. This guarantees that the hash can be replicated and validated on the blockchain and establishes a common method for encoding different kinds of data.

6. Put the atlas signature on the message: After that, the transaction index hash is translated to bytes, and the byte data is signed by the Ethereum wallet instance. When data is signed, a cryptographic signature is created that can be checked on the blockchain to confirm that the signer—the owner of the private key linked to the multisigAddress—signed the document.

7. Give the signature back: Ultimately, this signature is returned by the function. This signature acts as a cryptographic demonstration that the user has the private key linked to the wallet and that they have verified their identity using the proper OTP.

4.4 Database

I choose PostgreSQL as the database for my project mostly because of its reliability and capacity to manage secure transactions and complex search functions well. For applications that demand a high degree of security and dependability, PostgreSQL is known for complying to ACID standards, which ensures dependable transactions and consistent data integrity. Another important reason for wich i chose Postgres is because I had experience working with it.

Yet the PostgreSQL database structure in this project is very minimal, and this choice is directly tied to the fact that a large portion of the logic is implemented using smart contracts on the blockchain. By storing data and transactions directly on the blockchain through smart contracts, it can decentralize important activities and cut down on the need to use the database for the same things. For instance, blockchain is used to monitor and verify information about user transactions, contract statuses, and other important data, increasing security and transparency.

This means that the PostgreSQL database is used more for storing auxiliary data that does not require the same level of security or is not directly related to the logic of the smart contract. In this instance, all of the multisig wallets we've created on this platform are stored in the postgres database for quick access. This data is additionally on the blockchain, but we can store it in a database rather than continually searching the network once a wallet is configured and stops chaining. Database scalability is aided. It can also hold other data, such as user profile details like names, emails, preferred themes, and so forth, that is not included in the blockchain.

4.5 Frontend

For the frontend portion of the project, I decided to use Next.js due to a number of significant benefits this framework provides. I can use reusable components and state management to create a responsive and fluid user experience since Next.js integrates with React easily and is incredibly effective at creating modern, dynamic user interfaces.

The fact that Next.js can implement server-side rendering (SSR) and static site generation (SSG) is one of the key reasons I chose it. The information is pre-rendered and ready to be supplied to users and search engines, which enables me to optimize the application's performance and improve SEO and page load speeds. React server components allow for the direct fetching of data and waiting on it. As I mentioned before, this improves user experience and speeds up loading times by allowing viewers to see all of the data rather than just spinning images.

Additionally, Next.js has code optimization and automatic routing right out of the box, which cuts down on development time and the difficulty of maintaining dependencies and routes. These capabilities free me up from the minutiae of configuration and optimization so that I can concentrate more on creating application-specific functionality.

Folder Structure

The project uses a scalable folder structure in this project. I store all publicly accessible static resources (such photos, fonts, and favicon files) that may be directly referenced in code under the public directory.

The project's main section, the src directory, here are the following folders located:

- **app**: The Next.js project's app directory file structure is used by the App Router to directly define routes. Consider a file called app/page, for instance.The root route (/) naturally correlates to page.ts. Inside this folder you can also put different files like layout.ts, error.ts or loading.ts to handle different things like loading animations or error boundries.
- **components**: The components directory contains reusable React components for building the user interface. There are several components directories in other folders to organize components specific to each domain of the application, the top components directory is only used to store components that are used everywhere in the application
- **contracts**: The contracts directory is used to store contracts that are going to be used in the app like the multisig contract. It stores the contract ABI to make typescript understand how it can interact with the contract and make the development typesafe. It also stores the bytecode of the contract so the app can deploy instances of it.
- **db**: This directory contains everything related to database interactions, including configuration scripts, data models, and any other utilities needed to manipulate data.
- **helpers**: Utility functions that are utilized throughout the project are stored in the helpers directory. These aspects encourage the reuse of logic and prevent code duplication. The difference between the helpers and the lib directory is that the helpers does not contain any businiss logic specifically for this project.
- **hooks**: This is where React's custom hooks are placed, which allow state logic to be extracted and reused in components. This improves modularity, testability, readability and maintainablility.

- **lib**: The lib directory (short for "library") is used to store code that handles business logic or complex interactions in the application. For this project it is used to house the logic of the multisig and OTP.

- **service**: Services that manipulate data or perform specific operations are implemented in the service directory, acting as intermediaries between other apis or databases.

- **types**: this directory is used to define custom TypeScript types that are used to ensure correct typing within the project

- **validations**: This directory contains validation schemes necessary to ensure that the data entered into the application is correct and complete

Styling and building Reusable Components

I chose to use Tailwind CSS for its efficiency and flexibility. Tailwind provides utility classes that allow quick and direct HTML customization. This increases development speed and eliminates the need to write additional CSS code. Using granular design control, Tailwind helped me build responsive and custom interfaces with minimal effort. It was also extremely helpful in maintaining style consistency throughout the application.

Because it takes a lot of time to build components from scratch, I decided to use the NextUi and Shadcn UI libraries. Because you can import certain components, like buttons or modals, and not the complete library to use it, I choose to use two libraries. For instance, I utilized the modals from nextUi because I prefer their animations and they are more responsive for mobile devices, and I used the buttons from shadcn since they are much more simple.

Project Pages This are the folowing pages that are used inside the application:

- **Create Wallet**: This is the first page a new user sees. Here you can choose the chain, create a new multisig wallet, select the owners and the number of signatures required to approve a transaction

- **Select Wallet**: The page provides an interface to select one of the existing multisig wallets. It is useful for quickly switching between wallets.

- **Dashboard**: Here, users can see an overview of the selected wallet, including balance and other relevant metrics. It also provides quick access to frequently used functions, such as initiating a transaction or viewing your transaction history.

- **Atlas Page**: This is the page where you can setup your Atlas 2FA. This page has different states. One of the states is if atlas is not configured you can set it up with your authenticator. Another one is if Atlas is configured, than you can see details such as the atlas address.

- **Transactions**: This page lists all transactions made or proposed in the multi-sig wallet. Users can see a preview of how many owners have accepted the transaction as well as the history of all transactions that were initiated by this multisig.
- **Transaction Details**: You may view every detail of a particular transaction right here. The data field, value, transaction id, and transaction hash. If you are the owner, you can also validate Atlas and carry out the transaction if you have enough approvals. You can also approve or reject the transaction.

Chapter 5

Conclusions

Significant improvements in the safety and flexibility of cryptocurrency transactions can be achieved using Vault, a powerful solution for controlling the security of digital assets and cryptocurrencies on the Ethereum chain.

By requiring many owners to approve transactions, multisig, or multiple signature, increases security by lowering the risks involved in managing a single access point. Even in the event that a private key is compromised, this approach restricts the likelihood of financial theft and prohibits unauthorized access.

The Atlas feature of the application is among its most inventive features. This feature adds an extra degree of protection and verification before transactions are finalized, much like the two-step authentication (2FA) systems seen in web-based apps.

Because of its great versatility, the wallet may be configured in a variety of ways, from a single owner and approval level to simulating a 2FA wallet with Atlas capability. Users can adjust the security level to suit their own needs thanks to this. Additionally, money transfers require the 2FA code, even in the event that the single private key is compromised.

There are two primary areas that could be further addressed in terms of improvements. Since the contract already accepts ERC20 tokens, the first step would be to incorporate support for them directly into the application. By doing this, the wallet's functionality would be increased to support a larger variety of cryptocurrency assets. The second would be the development of a "Transaction Builder" page, such to what Safe provides [tra], that would enable users to construct intricate transactions in a clear and simple manner, hence improving accessibility and transaction management efficiency.

In summary, Vault stands out for its successful synthesis of cutting-edge security, originality, and adaptability. It also has a great deal of room for future growth to better serve user demands.

Bibliography

[arg] Argent. https://www.argent.xyz/.

[But14] Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform. *Ethereum.org*, 2014. Available online at https://ethereum.org/content/whitepaper/whitepaper-pdf/Ethereum_Whitepaper_-_Buterin_2014.pdf.

[eth] Etherscan api. https://etherscan.io/apis.

[har] Hardhat overview. https://hardhat.org/hardhat-runner/docs/getting-started#overview.

[inf] infura. https://www.infura.io/.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Bitcoin.org*, 2008. Available online at https://bitcoin.org/bitcoin.pdf;.

[NEF24] NEFTURE SECURITY I Blockchain Security. Private keys exploit, the most lucrative hack of 2023. https://medium.com/coinmonks/private-keys-exploit-the-most-lucrative-hack-of-2023-81390e0a29c
2024. Accessed: 2024-02-21.

[nexa] Image optimization. https://nextjs.org/docs/app/building-your-application/optimizing/images.

[nexb] Image optimization. https://nextjs.org/docs/app/building-your-application/data-fetching/server-actions-and-mutations.

[nexc] Route handlers. https://nextjs.org/docs/app/building-your-application/routing/route-handlers.

[rab] Rabby wallet. https://www.alchemy.com/dapps/rabby-wallet.

[rai] Rainbowkit. https://www.rainbowkit.com/.

[sdo] Solidity docs. `https://docs.soliditylang.org/en/v0.8.25/`.

[sma] What is a smart contract? `https://www.investopedia.com/terms/s/smart-contracts.asp/`.

[sola] Common solidity security vulnerabilities and how to avoid them. `https://metana.io/blog/common-solidity-security-vulnerabilities-how-to-avoid-them`.

[solb] Understanding solidity events and logs. `https://medium.com/coinmonks/understanding-solidity-events-and-logs-f29ea4f557cc#:~:text=They%20are%20written%20in%20Solidity,designed%20for%20the%20Ethereum%20platform.&text=Once%20the%20event%20has%20been,be%20accessed%20at%20any%20time./`.

[tai] Tailwind. `https://tailwindcss.com`.

[tra] Safe transaction builder. `https://app.safe.global/apps/open?safe=sep:0x5e18A2A9Fa4A99e8275A09b2125875F9c379B267&appUrl=https%3A%2F%2Fapps-portal.safe.global%2Ftx-builder`.

[typ] What is typescript? `https://www.typescriptlang.org/`.

[vie] viem. `https://viem.sh/`.

[wag] wagmi. `https://wagmi.sh/`.

[wat] How many blockchains are there? `how-many-blockchains-are-there`.