CODE OF DUTY

Exploring Self-Sovereign Identity

Literature Review

Index

Introduction

The Code of Duty “Exploring Self-Sovereign Identity” project required extensive research and insight into a variety of topics that are still relatively new and experimental, both to group members and in the technology field itself.

To bridge the gap in understanding and ensure successful, optimized implementation of the project, the group undertook research into the required fields, throughout the project.

This literature review will highlight the research and knowledge acquired to complete the project.

Further research into topics not specified in the initial requirements, is also included, to show the possible routes for expansion of the project and technologies therein.

Review of Sources

[1] An introduction to Blockchain technology is provided. The focus of the source is to establish a basic understanding of Blockchain and how it works. It also provides a guide on how to implement a private blockchain or “test chain” to perform transactions in a safe environment that simulates the functioning of an actual blockchain. The test chain in the source mimics the Ethereum blockchain and implementing the test chain privately would simulate using the actual Ethereum blockchain. General concepts relating to the blockchain is discussed. Terminology includes:

chainId: The private chain’s identifier used in replay protection.

homesteadBlock, eip155Block, eip158Block, byzantiumBlock: Relate to chain forking and versioning

difficulty: This dictates how difficult it is to mine a block. Setting this value low (~10–10000) is helpful in a private blockchain as it lets you mine blocks quickly, which equals fast transactions, and plenty of ETH to test with. For comparison, the Ethereum mainnet Genesis file defines a difficulty of 17179869184.

gasLimit: The total amount of gas that can be used in each block.

alloc: Allocates ETH to specific addresses.

[2] This source deals with states and transactions, how they are stored in Ethereum and how it is different from Bitcoin. More exploration of Ethereum’s data storage layer is done. The concept of blockchain “state” is introduced. The theory behind the Patricia Trie data structure is discussed and Ethereum’s concrete implementation of tries using Google’s leveldb database, is demonstrated. Bitcoin’s “state” is represented by its global collection of Unspent Transaction Outputs (UTXOs). The transfer of value in bitcoin is actioned through transactions. More specifically, a bitcoin user can spend one or more of their UTXOs by creating a transaction and adding one or more of their UTXOs as the transaction’s input. This model of UTXO makes Bitcoin different from Ethereum. Firstly, bitcoin UTXOs cannot be partially spent. If a bitcoin user spends 0.5 bitcoin (using their only UTXO which is worth 1 bitcoin) they have to deliberately self-address (send themselves) 0.5 bitcoin in return change. If they don’t send themselves change, they will lose the 0.5 bitcoin change to the bitcoin miner who mines their transaction. Secondly, at the most fundamental level, bitcoin does not maintain user account balances. With bitcoin, a user simply holds the private keys to one or more UTXO at any given point in time. Digital wallets make it seem like the bitcoin blockchain automatically stores and organizes user account balances and so forth. This is not the case. In contrast to the information above, the Ethereum world state is able to manage account balances, and more. The state of Ethereum is not an abstract concept. It is part of Ethereum’s base layer protocol. As the yellow paper mentions, Ethereum is a transaction-based “state” machine; a technology on which all transaction-based state machine concepts may be built. There are two vastly different types of data in

Ethereum; permanent data and ephemeral data. An example of permanent data would be a transaction. Once a transaction has been fully confirmed, it is recorded in the transaction trie; it is never altered. An example of ephemeral data would be the balance of a particular Ethereum account address. The balance of an account address is stored in the state trie and is altered whenever transactions against that particular account occur. It makes sense that permanent data, like mined transactions, and ephemeral data, like account balances, should be stored separately. Ethereum uses trie data structures to manage data.

[3] Smart contracts integration with Nethereum is discussed in the source. The purpose of this sample is the following: Understanding how to create contract deployment, function and event definitions to interact with a smart contract, creating an account object using a private key, allowing to sign transactions “offline”, deploying a smart contract (the sample provided is the standard ERC20 token contract), making a call to a smart contract (in this scenario get the balance of an account), sending a transaction to the smart contract (in this scenario transferring balance), estimating the gas cost of a contract transaction, Gas Price, Nonces and Sending Ether to smart contracts, signing online / offline transaction function messages and deployment messages, extension methods for Deployment and Function messages, retrieving the state of a smart contract from a previous block. Pre-Conditions: The ERC20 standard token contract is interacted with. The smart contract provides a standard way to create a new token, transfer it to another account and query the balance of any account. This standard interface allows the interoperability of smart contracts providing the same signature and applications that integrate with it. To deploy a contract one must create a class inheriting from the ContractDeploymentMessage, here compiled byte code and other constructor parameters can be included.The StandardToken deployment message includes the compiled bytecode of the ERC20 smart contract and the constructor parameter with the “totalSupply” of tokens. Each parameter is described with an attribute Parameter, including its name “totalSupply”, type “uint256” and order. Making a transfer will change the state of the blockchain, so in this scenario we will need to create a TransactionHandler using the TransferFunction definition. In the transfer message, one must include the receiver address “To”, and the “TokenAmount” to transfer. The final step is to send the request and wait for the receipt to be “mined” and included in the blockchain. Another option will be not to wait (poll) for the transaction to mined and just retrieve the transaction hash.

[4] Ganache is a personal blockchain for rapid Ethereum and Corda distributed application development. You can use Ganache across the entire development cycle; enabling you to develop, deploy, and test your dApps in a safe and deterministic environment. Ganache comes in two flavors: a UI and CLI. Ganache UI is a desktop application supporting both Ethereum and Corda technology. Our more robust command-line tool, ganache, is available for Ethereum development. It offers: console.log in Solidity, Zero-config Mainnet and testnet forking, Fork any Ethereum network without waiting to sync, Ethereum JSON-RPC support, Snapshot/revert state,

Mine blocks instantly, on demand, or at an interval, Fast-forward time, Impersonate any account (no private keys required!), Listens for JSON-RPC 2.0 requests over HTTP/WebSockets, Programmatic use in Node.js, Pending Transactions. All versions of Ganache are available for Windows, Mac, and Linux.

[5] Nethereum is the .Net integration library for Ethereum, simplifying smart contract management and interaction with Ethereum nodes whether they are public, like Geth Parity or private, like Quorum and Besu. Nethereum is being developed targeting netstandard 1.1, net451 and also as a portable library, hence it is compatible with all major operating systems (Windows, Linux, MacOS, Android and OSX) and has been tested on cloud, mobile, desktop, Xbox, hololens and windows IoT. Upcoming releases will be Ethereum 2.0 compliant (when Ethereum 2.0 is released) and include functionalities such as DevP2P, Plasma and Micro-Payments. Features: JSON RPC / IPC Ethereum core methods, Geth management API (admin, personal, debugging, miner), Parity management API, Quorum integration,Besu. Simplified smart contract interaction for deployment, function calling, transaction and event filtering and decoding of topics, Unity 3d Unity integration, Blockchain processing, ABI to .Net type encoding and decoding, including attribute-based for complex object deserialisation (nethereum-abi-encoding.md), Hd Wallet creation and management, Rules engine, HD Wallet integration, Transaction, RLP and message signing, verification and recovery of accounts, Libraries for standard contracts Token, ENS and Uport, Integrated TestRPC testing to simplify TDD and BDD (Specflow) development, Key storage using Web3 storage standard, compatible with Geth and Parity, Simplified account life cycle for both managed by third party client (personal) or stand-alone (signed transactions), Low level Interception of RPC calls, Code generation of smart contracts services.

[6] The video resource explores different ways of how to actively store data into the Ethereum blockchain using smart contracts written in solidity, with a focus on Data Types, Functions, Mapping & Structs.

[7] Ethereum is a decentralized, open-source blockchain network with Turing-complete smart contract functionality. Ether (ETH) is the native cryptocurrency. Users manage the global state of the Ethereum (execution layer) with a decentralized proof-of-work (PoW) consensus mechanism. The ETH2 (consensus layer) replaces PoW with a proof-of-stake (PoS) consensus mechanism. The Ethereum JSON-RPC API is a library of methods that interact with the Ethereum blockchain via JSON-RPC. Methods include functionality for reading and writing data to the network and executing smart contracts. Use the Ethereum JSON-RPC API on: Ethereum Mainnet, Ethereum testnets Rinkeby, Kovan, Görli, and Ropsten, Polygon, Optimism, Arbitrum. Transactions with type 0x0 are legacy transactions that use the transaction format existing before typed transactions were introduced in EIP-2718. They contain the parameters nonce, gasPrice, gasLimit, to, value, data, v, r, and s. Legacy transactions

don’t use access lists or incorporate EIP-1559 fee market changes. Access list transactions: Transactions with type 0x1 are transactions introduced in EIP-2930. They contain, along with the legacy parameters, an accessList parameter, which specifies an array of addresses and storage keys that the transaction plans to access (an access list). Access list transactions must specify an access list, and they don’t incorporate EIP-1559 fee market changes. EIP-1559 transactions: Transactions with type 0x2 are transactions introduced in EIP-1559, included in Ethereum's London fork. EIP-1559 addresses the network congestion and overpricing of transaction fees caused by the historical fee market, in which users send transactions specifying a gas price bid using the gasPrice parameter, and miners choose transactions with the highest bids. EIP-1559 transactions don’t specify gasPrice, and instead use an in-protocol, dynamically changing base fee per gas. At each block, the base fee per gas is adjusted to address network congestion as measured by a gas target. EIP-1559 transactions contain, along with the accessList parameter and legacy parameters except for gasPrice, a maxPriorityFeePerGas parameter, which specifies the maximum fee the sender is willing to pay per gas above the base fee (the maximum priority fee per gas), and a maxFeePerGas parameter, which specifies the maximum total fee (base fee + priority fee) the sender is willing to pay per gas. An EIP-1559 transaction always pays the base fee of the block it’s included in, and it pays a priority fee as priced by maxPriorityFeePerGas or, if the base fee per gas + maxPriorityFeePerGas exceeds maxFeePerGas, it pays a priority fee as priced by maxFeePerGas minus the base fee per gas. The base fee is burned, and the priority fee is paid to the miner that included the transaction. A transaction’s priority fee per gas incentivizes miners to include the transaction over other transactions with lower priority fees per gas.

[8] Introduction to Calls, Transactions, Events, Filters and Topics. The test contract: The following smart contract is an updated version of the “multiply” contract from the previous guide: The smart contract now includes an Event called “Multiplied”. The event will store on the log the original parameter for multiplication “a”, the address of the “sender” and the “result” of the multiplication. The parameter “a” and the “sender” address are both indexed so we can create specific filters for those two values using topics. Deploying the contract: As per the previous guide, we can deploy the contract. The multiply transaction: When performing a call, we are either retrieving data which is stored in the smart contract state or we are performing an action (i.e multiplication), calls are not transactions which are verified through the blockchain consensus. Submitting a transaction to perform a function operation in a smart contract does not return the result of the operation, events can be used to retrieve information or we can inspect the state of the smart contract by using function calls. Using the contract address from deploying the transaction we can create an instance of a contract object and the function “multiply”. The function object simplifies submitting transactions in the same way as calls. As per the example above we just need to include the “senderAddress” which will be charged the gas associated with the operation together with the parameters for the function operation. There is also the option to specify the gas or include an Ether value as part of the transaction. On the example, we have

submitted 2 transactions to perform a multiplication for 7 and 8 respectively, and are waiting for the transaction to be mined on our private test chain. Events, filters and topics. Creating events and filters: Events are defined as part of the abi, and similarly to the functions we can get events using our contract instance. The event object allows to create filters in order to retrieve the information stored on the log. In the example above we are retrieving the logs in which the multiply parameter is 7, because the input parameter for the multiplication is marked as indexed, we can filter for that topic. In a similar way, we can filter the sender address as it is also marked as indexed, but if we wanted to filter for that specific topic we will use the second parameter when creating the filter. Event DTO: Event data transfer objects allows to simply decode all the event parameters into a transfer object, in a similar way as we will deserialise a Json object. In the example above, the MultipliedEvent properties have been “mapped” with custom parameter attributes to the event parameters. Each parameter specifies the original type, name, order and if is indexed or not. As we can see, types like address are decoded into strings and in our scenario we are safe to decode int256 to int32 but if not known, the final type BigInteger would have been a better option. Retrieving the events and logs: Using the filters we have already created, we can retrieve the logs and events. Above we are using GetFilterChanges, which can be used to retrieve any logs that matches our criteria since the filter was created or since the last time we tried to retrieve the changes. Other option would have been to use GetAllChanges using the FilterInput.

[9] The Netherium Playground provides a test environment to simulate Blockchain transactions and helped form the basic and understanding of the implementation of the blockchain in the project itself.

[10] A technical guide on how to get Ethereum’s transaction details from its transaction hash, with code examples and further exploration into the uses thereof.

[11] A look at Microsoft Azure’s “Key Vault.” Enhance data protection and compliance. Secure key management is essential to protect data in the cloud. Use Azure Key Vault to encrypt keys and small secrets like passwords that use keys stored in hardware security modules (HSMs). For more assurance, import or generate keys in HSMs, and Microsoft processes your keys in FIPS validated HSMs (hardware and firmware) - FIPS 140-2 Level 2 for vaults and FIPS 140-2 Level 3 for HSM pools. With Key Vault, Microsoft doesn’t see or extract your keys. Monitor and audit your key use with Azure logging—pipe logs into Azure HDInsight or your security information and event management (SIEM) solution for more analysis and threat detection. All of the control, none of the work. Use Key Vault and you don’t need to provision, configure, patch, and maintain HSMs and key management software. Provision new vaults and keys (or import keys from your own HSMs) in minutes and centrally manage keys, secrets, and policies. You keep control over your keys—simply grant permission for your own and partner applications to use them as needed. Applications never have direct access to

keys. Developers manage keys used for Dev/Test and seamlessly migrate to production the keys that are managed by security operations. Simplify and automate tasks related to SSL/TLS certificates—Key Vault enables you to enroll and automatically renew certificates from supported public Certificate Authorities. Boost performance and achieve global scale. Improve performance and reduce the latency of your cloud applications by storing cryptographic keys in the cloud, instead of on-premises. Key Vault quickly scales to meet the cryptographic needs of your cloud applications and match peak demand, without the cost of deploying dedicated HSMs. Achieve global redundancy by provisioning vaults in Azure global datacenters—keep a copy in your own HSMs for more durability.

[12] An introduction and documentation for Three.js, a framework used for three-dimensional graphics and animations in JavaScript. This forms part of the research when deciding how to implement the “Avatar” functionality of the SSI system.

[13] Remix Ethereum provides a test environment to simulate Blockchain transactions and helped form the basic and understanding of the implementation of the blockchain in the project itself. It serves as an online IDE for blockchain interactions.

Conclusion

The research detailed in this review ensured that every group member was well-informed on the subject matter, to the extent where we could apply concepts learned in new and experimental ways, that fit our needs for the Self-Sovereign Identity Project. The knowledge gained from the research informed important project decisions in terms of technology used and specific implementation details. By conducting this research, we were able to ensure that the technology used is relevant, optimal and well-suited to all the requirements and specifications.

References

1. Mercury Protocol. 2017. Available from: https://medium.com/mercuryprotocol/how-to-create-your-own-private-ethereum-blockchain-dad6af82fc9f [Accessed: 11 July 2022]
2. Vasa. 2018. Available from: https://medium.com/hackernoon/getting-deep-into-ethereum-how-data-is-stored-in-ethereum-e3f669d96033 [Accessed: 11 July 2022]
3. Netherium Documentation. 2022. Available from: https://docs.nethereum.com/en/latest/nethereum-smartcontrats-gettingstarted/ [Accessed: 12 July 2022]
4. Truffle Suite. 2022. Available from: https://trufflesuite.com/ganache/ [Accessed: 14 July 2022]
5. Netherium. 2022. Available from: https://nethereum.com/ [Accessed: 14 July 2022]
6. Cryptological. 2018. Available from: https://youtu.be/ZkWaAKl8npU [Accessed: 15 July 2022]
7. Infura Inc. 2022. Available from: https://infura.io/ [Accessed: 24 July 2022]
8. Netherium Documentation. 2022. Available from: https://docs.nethereum.com/en/latest/contracts/calling-transactions-events/ [Accessed: 25 July 2022]
9. Netherium Playground. 2022. Available from: http://playground.nethereum.com/csharp/id/1007] [Accessed: 27 July 2022]
10. Moriya, H. 2018, Available from: https://piyopiyo.medium.com/how-to-get-ethereums-transaction-details-from-its-transaction-hash-b6a376887491 [Accessed: 28 July 2022]
11. Microsoft Azure. 2022. Available from: https://azure.microsoft.com/en-us/services/key-vault/ [Accessed: 1 August 2022]
12. Three.js. 2022. Available from: https://threejs.org/ [Accessed: 1 August 2022]
13. Remix Ethereum. 2022. Available from: https://remix.ethereum.org/ [Accessed: 21 August 2022]