UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Aleksandr Tsõganov

Integrating User Identity with Ethereum Smart Contract Wallet

Master's Thesis (30 ECTS)

Supervisor(s):
Orlenys López Pintado
Aivo Kalu
Kristjan Kuhi

Tartu 2019

Integrating User Identity to Ethereum Smart Contract Wallet

Abstract:
The first major application of the blockchain technology was made for cryptocurrencies and by now it is used in numerous industries, including in energy, agriculture, manufacturing, etc. The original idea of transferring assets from one account to another has to be updated for those industries. Non-financial industries have a different definition of an asset and a differing attitude towards the anonymity of the users, i.e. it is necessary for the users and their wallets to become more public.
Namely, the main problem is related to the users' anonymity and uncontrolled asset transfers in decentralized applications. In this thesis, the user's identity is connected with his blockchain wallet to allow asset transfers to take place only with added identity-based signatures of the approver and the user himself. The implementation of the thesis includes the analysis of the Ethereum blockchain principles, different wallet protection solutions and state-level identity services. The thesis proposes a specification of an identity-based wallet integration with Dapp. The solution specification is validated using Dapp and a smart-contract prototype.

Keywords:
Blockchain, smart contracts, Ethereum, digital signature, user identity, Smart-ID

CERCS:
T120 Systems engineering, computer technology

Kasutaja Identiteedi Integreerimine Ethereum Nutika Lepingu Rahakotiga

Lühikokkuvõte:
Esimene suurem rakendusplokiahela tehnoloogias oli krüptovaluuta ja selle vahendamine, praeguseks on aga plokiahela tehnoloogia leidnud kasutust paljudes teistes tööstusvaldkondades nagu energeetika, põllumajandus, tootmine jt. Algne idee, mis hõlmas varade saatmist ühelt anonüümselt kontolt teisele, vajab uuendusi lähtuvalt uute valdkondade vajadustest. Mittefinantssektorites võib vara määratlus olla erinev ning suhtumine kasutajate anonüümsusesse samuti s.t, et kasutaja ja tema rahakott muutuvad sellisel juhul avalikumaks.
Peamine probleem seisneb kasutaja anonüümsuses ja varade saatmise üle kontrolli puudumises. Antud lõputöös me ühendame kasutaja identiteedi tema plokiahela rahakotiga selleks, et lubada varade saatmist alles peale digitaalset signeerimist kinnitaja ja kasutaja enda poolt. Lõputöö käigus analüüsitakse Ethereum plokiahela põhimõtteid, erinevaid plokiahela rahakoti kaitselahendusi ja riigi poolt väljastatud identiteedil baseeruvad e-identimise teenuseid. Lõputöö tulemusena esitatakse identiteedi põhise rahakoti ja detsentraliseeritud rakenduse integreerimise spetsifikatsiooni. Töö tulemuse valideerimiseks on kasutatud prototüüpi detsentraliseeritud rakendusest ja plokiahela nutilepingust.

Märksõnad:
Plokiahel, nutikas leping, Ethereum, digitaalne signeerimine, kasutaja identiteet, Smart-ID

CERCS:
T120 Süsteemitehnoloogia, arvutitehnoloogia

Table of Contents

Acknowledgments

I would like to thank WePower Network that gave an opportunity to do the research that gives new inputs to their software system development. Additionally, Cybernetica and Catapult Labs for giving advice on technical parts and my supervisors for their advice and recommendations.

1 Introduction

Blockchain technologies are decentralized systems, originally developed for transferring digital currencies. Currently, there are many potential applications of this technology in different kinds of industries, for example, energy market agreements and agriculture supply chain (Al-Jaroodi & Mohamed, 2019). Blockchain technology is gaining use in such industries because of its tamper resistance, making hacking practically impossible. In case of currency transactions, users regard anonymity of public blockchain systems as one of the key characteristics to avoid being publicly linked to their assets for safety reasons. Assets in blockchain are stored in wallets, which are protected using users' private keys. As private keys give control over wallets, there are many solutions developed to secure private keys. These solutions have usability problems, for example USB based solutions require a specific software and a computer to access the wallet. In addition, very few of them connect the blockchain wallet to users' identity, which makes it difficult for companies using blockchain technologies to support their customers in case of any issues.

In one hand, anonymity is important for safety reasons and, in the other hand, platforms that are using public blockchain technology as part of their system are facing difficulties with supporting and protecting their users' interests due to the anonymity of the latter. In non-financial industries, where blockchain is used due to its decentralization, persistency and auditability characteristics, the definition of an asset can be different. In such cases, the fact that the users and their wallets become more public, because their tradable assets have merely a prospective promissory value, may not constitute a problem in itself. Also, these solutions may have no possibility to transfer the referred assets with just having access to the wallet, because additional approvals of another user or service are required. To implement that, a solution is required for identifying users to enable multiple-step approvals or connections to such solutions.

One of such solutions is required by WePower Network, which has a platform that uses an Ethereum blockchain decentralized application. There is a need to have users with verified identity on this platform to provide trust for electricity purchase agreements on the platform. To operate on state level in Estonia, it is required to have the identity verification service on the same level as is performed by the state-issued digital identity verification service provider in Estonia. The decentralized application itself is using Ethereum blockchain wallets to make asset transactions between the users. The assets themselves contain agreements between the users of the platform.

To give a better understanding of the problem context we will go through a simple example visualized in Figure 1. The problem originates from the electricity domain, which might be hard to understand. To make it somewhat clearer and simpler, we will explain electricity through the example of water. Provided that we have an app that holds agreements between A and B during a specified period of time. Let us assume that this contains a daily drinking water supply agreement and there is lack of water in the state. We have a policy that limits the amount of water for each participant, for example to three buckets for participant A. The participant B is a producer of drinking water and needs to supply all the produced water each day, otherwise having to pay a penalty due to regulatory rules. The app needs to verify that all of the produced water from B is covered by the supply agreement, so that there is no waste nor over-deliveries. To ensure auditability and storage of data in a distributed way across the globe, the app uses Ethereum blockchain. The aim of the application is to sustain control over the water supply agreements and ensure that the participants will not be breaching applicable policies. If agreements were made between two participants only with no third-party verification it would be difficult to guarantee that the agreements did not

violate the relevant policies. However, in case of under- or over-delivery (i.e. a breach of the agreement), the enforcement of the aggrieved party's rights would be hindered due to the agreement having been signed in blockchain where users are anonymous.

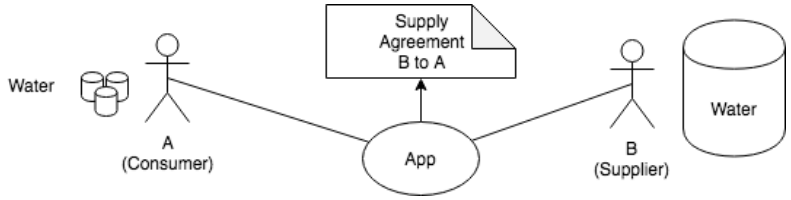

Figure 1 Blockchain application with agreement assets

The goal of this thesis is to propose a solution that ensures user wallet security and gives additional value to decentralized applications by identifying anonymous users with verified identity. An additional goal is to allow the transfer of assets between users with the acceptance of the decentralized application approver signature.

The scope of the thesis is an Ethereum blockchain based application, the integration of user's identity verification and signing service and the development of approver functionality. The main question is how to connect the user's identity to the wallet in a way that transactions can be made only after identity-based signature verification. For more detailed answer we separated the main question into sub questions:

- SQ1 How to make the decentralized application know who is making the transactions?
- SQ2 How to associate an asset transfer with the user's identity?
- SQ3 How to integrate the state-level identity-based signing into the Ethereum blockchain?
- SQ4 How to create a wallet with multiple transaction approval steps?
- SQ5 How to verify identity-based signatures in Ethereum blockchain?
- SQ6 How to check that a transaction made in the past, was made with valid identity?

This thesis consists of five chapters:

Chapter 1 presents the topic, scope, problem and questions that the thesis will solve.

Chapter 2 includes the review of the literature of the problem space in three sections:

a. The Ethereum Blockchain is explored to gain an understanding of what its core mechanism is and where the solution would be integrated.
b. The current solutions available for Ethereum wallet protection are reviewed and compared to understand what approaches are used in current solutions and what are the differences between them.
c. The review of the state-level identity verification services available in Estonia.

This chapter also presents the summary of the literature review and analyzes the possible application of the knowledge gained from the literature to the proposed solution.

Chapter 3 explains the requirements and current state to create the specification for identity-based wallet integration with Dapp. It is a more detailed analysis of the results and choices made in Chapter 2, with real examples.

Chapter 4 presents the prototype of the solution to the problem outlined in this thesis. This chapter also validates the prototype against the specification.

Chapter 5 summarizes the work, presents the solution and outcomes and provides concluding remarks.

2 Background and literature review

This chapter covers the problem space and evaluates the working solutions that need to be taken into account in the development of the one solution presented later in this thesis.

The problem space is Ethereum Dapp[1], a blockchain based application that uses its own currency tokens[2]. These tokens are transferred from user to user and everything is written into the blockchain. Because the solution needs to integrate with blockchain, the Ethereum blockchain functionality is described in Section 2.1. A wallet is a part of the blockchain and hence also a part of the problem space. To answer SQ1 and SQ4 research questions, security solutions for wallets are reviewed and compared in section 2.2. Based on the requirement of using the Estonian national identity services, the available possibilities for integration were reviewed and compared in section 2.3 to support answers for SQ2, SQ3 and SQ5.

2.1 Ethereum Blockchain

Blockchain is a technology that is based on a digitally distributed ledger system that has no central storage owner authority. Unlike the traditional cloud storage approaches, the Ethereum blockchain technology is open, which means that all records are trustfully shared among parties. All users who run a certain blockchain software are called nodes. Nodes store individual copies of records of the entire blockchain system. As nodes can be located anywhere in the world, blockchain systems are globally decentralized. Because everything stored in nodes is public, the system is self-verifying being in a so-called state of consensus. In addition to storing, nodes carry out a verification process of new data validity against other nodes' copies (also called the mining process). Such process enables the blockchain to be robust, which means that all the data stored in the network cannot be manipulated by a single node and has no single point of failure. Blockchain networks are transparent and incorruptible, which makes them safe to be called the single source of truth (RF Wireless World, 2012) (Blockgeeks, 2018) (Berryhill, Bourgery, & Hanson, 2018).

Ethereum is an open source platform that runs on the Ethereum Virtual Machine (EVM). EVM is a turing complete software that is working in the Ethereum network. An abstract layer of the network enables to create own rules of ownership, transaction formats, and state transition functions. These features are possible due to smart contracts, which are a set of rules that are executed only if certain conditions are met (Vujičić, Jagodić, & Ranđić, 2018). In addition, this solution enabled the possibility to create decentralized applications. Ethereum provides its own high-level programming language named Solidity. Solidity is one of the most common languages in blockchain solutions for writing smart contracts, which is very similar to JavaScript. Solidity enables engineers to develop registries of debts or promises, different markets or move assets according to some instructions as defined in the past (Grech & Camilleri, 2017) (Destefanis, Marchesi, & Ortu, 2018).

[1] Dapp – decentralized application of Ethereum platform.

[2] Token – Ethereum platform can issue tokens for Dapps. Tokens are smart contracts that implement standards of Ethereum. In majority of Dapps, the ERC20 Token Standard is used as the native currency (Vujičić, Jagodić, & Ranđić, 2018).

Blocks

Blocks in Ethereum Blockchain contain a block number, difficulty, nonce[3], transaction list and the most recent state. The blocks are containers of smaller fractions of data – the transactions. To enable data validation across the network each block has a hashed link to the previous block, as shown in Figure 2. The hashing process is a mathematical problem that must be mined by the computational power of a node in the network. The output of the hashing process obtained by the input values passed through a cryptographic algorithm results in a unique combination of numbers and letters with a fixed number of bits (Berryhill, Bourgery, & Hanson, 2018). The hashed links enable blockchain data to be immutable and consistent across the network. There is no way to remove data that has been written into the blockchain. Distributed network is about consistency across all of the participants in the blockchain network. Ethereum uses a proof-of-work validation scheme where all participants must agree on a common ledger and have access to all entries of blocks ever recorded (Valenta & Sandner, 2017).

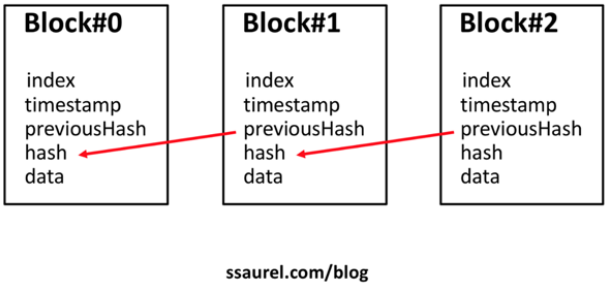

Figure 2. Blockchain block link visualization (Saurel, 2018)

Transactions

A blockchain transaction is the core information that is stored as data in the blocks. Creating a smart contract, broadcasting a smart contract and interacting with it is done by separate transactions. Basically, one type of interaction with a smart contract can be described as a banking transaction between persons A and B that is written in a hashed format. The transaction flow (Wei, 2017) of the Ethereum blockchain works as follows

1. The person A is requesting a transaction to the person B.
2. The transaction is digitally signed and broadcasted to the network for validation according to a logic, which is set by the creators of the Ethereum blockchain system. In a transaction, there can be one or many inputs and outputs.
3. One node validates the broadcasted transaction and picks a block to include the processed transaction in. When the transaction is marked as valid and stored in the block it will wait until the prerequisites of a complete block are fulfilled.
4. The fulfilled block is sealed with the hash as described before and distributed across the network to validate the new block with its hashed value.
5. The block is published to the network once all the nodes are confirming the new block's hash.

[3] Nonce – the number of transactions sent from a given address.

Transaction Costs

Each transaction made on Ethereum blockchain costs a certain amount of so-called "Gas". The sum of the Gas cost is calculated as sum of every operation performed in a transaction. Each block has a Gas limit, the amount of execution work that is needed to perform a transaction and pay for that execution process itself. Ethereum Virtual Machine spends a certain amount of Gas during the transaction execution according to specific rules (Ethereum, 2019).

The Gas parameter has its own price, which is configurable by the transaction creator. The Gas limit is multiplied by the Gas price and paid upfront before the execution of the transaction. If some Gas is left after the execution, it will be refunded to the creator of the transaction. The Ethereum network has limits to the transaction processing speed based on the nodes' availability. The less is paid for the Gas the longer it takes to perform a transaction. If more Gas is needed for the transaction than allocated, the process is fully reverted to the beginning state (Ethereum, 2019).

Wallet

Transactions in a blockchain can be made using specific assets, for example a numeric value in a Dapp currency, described in a blockchain system, it can be a digital currency, energy tokens or any value that can be transferred between the participants of the blockchain. In most of the blockchain systems the information on these assets is presented in a hashed form. To make transactions, there is a need to have a wallet, which includes these assets. A blockchain wallet is an asset account, that has two types of keys – public and private, both generated when the wallet is created (Andreas M. Antonopoulos, 2018).

Public and private keys

A public key is a public piece of information in a blockchain network, it is similar to a bank account number in a bank. Making transactions in a blockchain system can be described as sending assets from public key A to public key B. Public keys are created using cryptographical algorithms. Special properties in the mathematical functions used in these algorithms makes the creation of such keys simple, but also make it very hard to calculate their inverse (Berryhill, Bourgery, & Hanson, 2018).

A private key is used to sign the transactions and is the only way to access the digital assets in the wallet. Although the blockchain system itself may be secure enough, the security of the private key has to be ensured by the users themselves (Andreas M. Antonopoulos, 2018).

Smart Contracts

When the basic implementation of a distributed ledger was introduced in Blockchain 1.0, cryptocurrency became the most common application of blockchains. The financial market was the simplest way to popularize blockchain systems. Large amounts of digital currencies were implemented for trading and investing. The next step was the implementation of the Ethereum Blockchain, in Blockchain 2.0, which introduced the smart contract concept (Berryhill, Bourgery, & Hanson, 2018) (Unibright.io, 2017).

Smart contracts are a set of rules that can be executed when conditions are met, they are similar to the users of the blockchain network. These are pieces of written scripts that are

connected to the blockchain network. The main advantage of smart contracts is that these are nearly impossible to hack, because they are controlled by the blockchain logic, which reduces the costs of execution and verification and also protects from fraud or third-party interference (Berryhill, Bourgery, & Hanson, 2018).

Because smart contracts can be written by anyone, it enables individuals and companies to write some of the business logic directly into the blockchain. The simplest example of a smart contract is user A sending assets to user B after, for example, 5 days. The smart contract definition describes a time-based condition, takes assets from A, holds them as a third-party user in the network and then allows to execute the transaction to B only when 5 days have passed. Smart contracts enable to create decentralized autonomous organizations in the blockchain systems (Christidis, 2016) (Berryhill, Bourgery, & Hanson, 2018).

In more detail, the smart contract can be separated into four independent parts: contractual arrangements, control of preconditions, execution and finalization (Figure 3). In the contractual arrangements (also called "create" activity), the parties who are involved decide on their liabilities and write these into the code. As an output, there is an executable form stored in the blockchain system. In the preconditions phase (known as "freeze" activity), every node in the blockchain system can assess the smart contract to check the conditions, so that the written logic is stored on the blockchain. This is the governance part of a smart contract as it is checked publicly on the blockchain. It is followed by the execution part, which evaluates the transactions that were defined to be executed in case the conditions are met (Christian Sillaber, 2017).

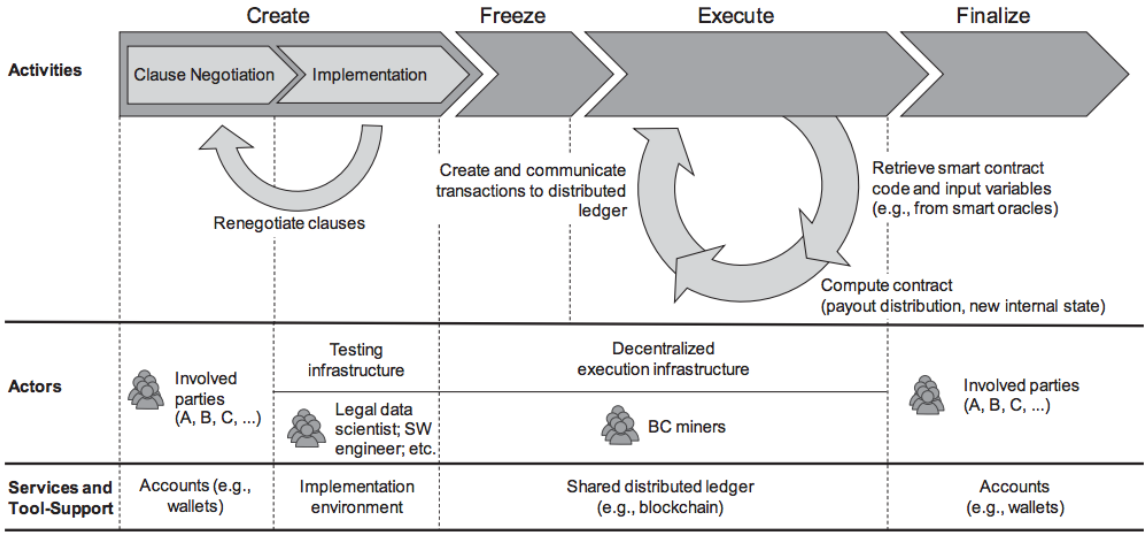

Figure 3 The life cycle of a smart contract: phases, actors and services (Murthy, 2017).

2.2 Wallet protection solutions

The literature review has produced a feature overview of existing solutions, that were found in the reviewed sources. The features are listed in Table 1. The main purpose is to analyze the different solutions available for protecting blockchain users' asset wallets and their connection to the users themselves. To get a better overview, solutions made for different Blockchains have been covered. There are many technologies and methods to protect the

user's wallet, each of these having specific pros and cons. Most blockchain systems are using similar solutions, but Ethereum for example, has additional solutions due to smart contracts (Cheetah Mobile Blockchain Research Lab & Cheetah Lab, 2018).

Table 1 compares five different solutions that are being used for wallet private key storage and/or for security enhancement. For the purposes of answering SQ1, two possible types of wallet security were discovered – hardware-based and software-based. As wallet security is a part of the blockchain process that requires user interactions, it leads directly to SQ2. The aim is to find a solution that requires a smaller number of human actions, a possibility to connect the user with his identity and allow the introduction of an additional approval step before transferring assets.

Table 1 Comparison of wallet protection solutions

Name	**Type**	**Protection**	**Connectivity technology**	**Connects to**	**Approval possibility**	**User identity**	**Backup possibility**
BlueWallet	Hardware	PIN	Bluetooth	Mobile; computer	no	no	unknown
Ledger	Hardware	PIN 4-8 length	USB	Computer	no	no	yes
Multisignature	Smart-contract	Relies to code and contract owner wallet security	Blockchain network	Depends on last security method applied	yes	no	yes
MetaMask Wallet	Browser-extension	User password. 12-word recovery	Internet, browser-plugin	Computer	no	no	yes
Coinbase	Web-Service	Username, password, 2-step Auth	Internet	Anywhere on the web	no	no	yes

The following sections explain the wallet protection solutions from Table 1 in more detail.

2.2.1 Ledger

A ledger is a hardware wallet created specifically for protecting blockchain private keys. It stores the private key entirely on hardware and provides access to it when a PIN-code is entered. The initial setup of ledgers includes a recovery code generation to allow the private key recovery in case of device damage or other issues. To allow for the recovery, it is necessary to securely save the number of words that are needed to be entered in case of restoring the key. The device can store 25 different blockchain private keys at the same time (Ledger, 2018) (Bruno, 2017).

The Ledger solution is one of the simplest and the most popular one. The risk remains in the software failure of the ledger device or in its physical damage. There are also usability issues associated with the need of using a computer and a connecting wire each time a transaction is made.

2.2.2 BlueWallet

A BlueWallet is a hardware solution that is implemented to securely authorize blockchain transactions. It is mainly focused on Bitcoin blockchain transaction verification. The BlueWallet represents a proof of concept of secure communication between hardware and blockchain networks using a Bluetooth Low Energy connected device. In addition, this solution can be used as an alternative payment method instead of using cash or credit cards (Bamert T., 2014).

This concept requires that transactions that are made in a Bitcoin Blockchain can be prepared offline from the blockchain network. This allows the solution to be completely separated from any network. The transaction should be confirmed by the user's private key, which is safely stored in the hardware itself and protected with a PIN code. In case a bitcoin transaction is made using a computer, it allows to sign the transaction using a Bluetooth connected BlueWallet hardware for enhanced transaction security (Bamert T., 2014).

This approach uses quite a simple solution for storing a private key on hardware, but in addition, it provides a convenient mechanism to sign transactions wirelessly, while traditional hardware solutions are wired. The main risk of theft is reduced, since the user is disconnected from the internet, although hardware errors or physical problems are still possible. The access to the private key is simplified to having only a PIN code, which increases the risk in case the device is stolen.

2.2.3 Multisignature

The Multisignature wallet also referred to as "multi-sig" is a different approach to solving the private key security issue. Multisignature allows to create multiple private keys that need to be used for signing a transaction. In this approach, multiple users have independent private keys, which are applied as authorization signatures in a Multisignature wallet. This solution requires more than one individual to use his key to make a transaction from the wallet. For example, if you are having a multi-sig wallet in a three-person family, then if something happens with one of the family members the wallet can be accessed by two others (Hannan, 2018) (Unchained Capital, 2018).

The control remains in the user's hands, so although it is better to have multiple sources and possibilities, this solution is still subject to human error. However, with this solution the recovery of a stolen key is possible. Technically, to make Multisignature work, it is required to have smart contracts in the blockchain. Each transaction that will be made using a Multisignature wallet should be executed within the framework of specific smart contract(s). It adds complexity to the system and may cause vulnerability in case of an incorrect implementation. This means that for each transaction there is a need for multiple users and this solution may not be as user-friendly as the ones that rely on a single user (Hannan, 2018) (Unchained Capital, 2018).

2.2.4 MetaMask

MetaMask is a web browser extension wallet that works for Ethereum blockchain. It is an open source project that creates connections between Chrome, Firefox and Brave browsers and Ethereum blockchain. This solution uses *Hierarchical deterministic settings* by giving the user a list of words that can be used to reset lost account information. MetaMask allows

users to store their private and public keys in their own browser's local storage. This solution is not securing the keys completely, because while it is reasonably safe to store your own keys in the browser application, it also has a connection to the internet, which presents other security issues. While the browser is connected to the internet, there is a risk of facing *phishing attacks* using browser's opened tabs and stealing information through fake forms or pop-ups opened from websites (King, 2018).

MetaMask improves the usability of private key in terms of ease of transactions, but it makes the user's computer and its browser susceptible to malicious actions that can be made through the Internet. The list of words for restoring access to the keys needs to be stored in some place physically or electronically, which is nearly the same level of risk as storing the private key itself.

2.2.5 Coinbase Wallet

Coinbase is a web-based digital currency market that offers its own wallet storage. The user's wallet keys data is stored using AES-256 encryption on Coinbase servers. The servers are entirely disconnected from the Internet for extra security. A secret sharing method is used to split the data and backup on USB drives and papers, which are geographically distributed around the world. For a user, there is both a username and password protection as well as a 2-step verification available. Coinbase fully manages the private key of the user's wallet, so the user cannot use or even see the key separately (Coinbase, 2019). The service is widely used, which means that this web-service wallet solution is trustworthy and widely accepted by the users. Coinbase has more than 25 million users by today (Wilma Woo, 2018).

The usability is quite good, due to the private key management by Coinbase. There is an application available to interact with the wallet. The risk is mostly on the service-side to securely hold and use the private key. User only stands for his account password and 2-step verification process, which is quite simple and recoverable. This solution does not provide identity verification.

2.2.6 Comparison

Ledger is a solution that does not connect to internet, so we cannot access and verify user's identity online. This is needed for our solution, because identity related services have certificates that need to be checked for validity. It has a closed hardware, that cannot be modified and integrated with identity. Interaction needs a computer, which means solution usage on mobile device is not possible. BlueWallet is nearly the same, except that it is wireless.

MetaMask is connected to computers browser and has no connection to user identity that is also needed for our solution. Additional approving step is not allowed.

Coinbase web-service Wallet solution is a good example to use for implementing own solution, because it has a large number of users that supports, and trust in Coinbase web-service. In that field number of users means that service is trustful and well working. The wallet can be accessed using internet from any device, but there is no possibility to develop own integrations, so identity solutions cannot be connected.

Multisignature allows to use example script that is deployed to the blockchain network for usage. As it is open-sourced it can be taken in to account as a template to make own

modifications. Cons of Multisignature is that it is fully self-written and it can be not well tested or secured against issues. Multisignature is not device related, because it is in the Blockchain network. Multisignature wallet can be controlled using Dapp.

2.3 User identity verification and signature services

This section briefly describes three possible ways of electronic identity (eID) verifying at state level in Estonia. The aim of this section is to point out the specifications that will help answer questions SQ2 and SQ3. All of the services use certificates issued by SK ID Solutions AS which also provides validity confirmation and time-stamping services. First solution is based on national ID-card, second and third are mobile services. An overview of the solutions and comparison of the features and prices are compared (SK ID Solutions AS, 2019).

Table 2 gives an overview of the three Estonian nation-level identity services and their differences. The important point was the operational complexity, from the user's perspective, of any action that requires confirmation of identity. Number of users is showed to get overview of potential use of the service. Prices are taken into account to see approximate cost of solution and additional charge for user.

Table 2 Table concluding different Identity Services

Service	ID-card based	Mobile-ID based	Smart-ID based
Price for user	25€ citizen Adult or 100€ for	1€/month	0
Number of steps for authentication	4	3	3
Equipment needed for use	ID-card, computer, Internet, card reader	Mobile phone, mobile phone network connection	Smartphone (Android/iOS), any kind of internet connection
Number of access points per identity	1	1	More than one (limit not specified)
Number of users in Estonia	~1198876 (Estonia, 2019)	~193000 (Sibold, 2018)	~341803 (Lõugas, 2018)
Number of users in total	~1198876 (Estonia, 2019)	~400000 (SK ID Solutions AS, 2019)	~2001187 (Smart-ID, 2019)
Integration layer	SOAP API DigiDocService	SOAP API DigiDocService	Smart-ID REST API

2.3.1 ID-card based solution

ID-card is a mandatory identity document in Estonia which is connected to one person's physical identity. It is a secure data carrier for keys and certificates to verify user's identity. The card itself can be used for many operations, e.g. as a national ID document, national health insurance card, proof of identification for bank accounts, i-Voting, for digital signature creation and to access many other e-services (Republic of Estonia Information System Authority, 2019; e-estonia, 2019).

ID-card has a chip which is used to read certificates from the card. The public key infrastructure (PKI) model uses secret and public keys which are stored separately. The connection of the keys is used to prove person's identity for e-services or to make a digital signature. The secret key is protected and can only be used by its original owner and the public key is available to everyone. All of the operations that can be performed using an ID-card are PIN-protected to prevent misuse of the card (Republic of Estonia Information System Authority, 2019).

To use this service there is a need for the government issued card itself, which can be issued to Estonian citizens or e-Residents (Politsei- ja Piirivalveamet, 2019). In addition to the card itself there are also issued PINs which are needed to perform any operation with using the card.

Each card has certificates stored inside that are needed to be valid and up to date to perform any digital actions. Certificates are validated every time they are used for authentication or document signing procedure against Certificate Authority service. This validation service should be used by e-service provider (SK ID Solutions AS, 2018).

Computer with an internet connection, card reader and ID-card software are needed as additional equipment to use the card-based services (id.ee, 2019). The process of authentication for e-service includes 4 steps:

- Connecting to computer using a card reader
- Entering e-service
- Requesting authentication using an ID-card
- Entering the card's PIN code

2.3.2 Mobile-ID based solution

Mobile devices are the most used personal devices in these days and mobile based electronic identity services are becoming widely used around European Union. Mobile-ID service is an addition to Estonian ID-card based solution, which uses special SIM card to perform eID authentication and signature creation. The service has to be ordered from the mobile network operator and costs 1€ per month (Kubach, 2015).

Issuing Mobile-ID SIM card follows the same requirements as an ID-card. The user's private keys are stored on the SIM card's chip and are only accessible with valid PIN codes that are provided by the issuer. SIM card has a small application that is used to deliver authentication and signature processes through the mobile network. All data exchanges over the service are performed over an encrypted connection (e-estonia, 2019).

To integrate an e-service with Mobile-ID there is a need to use DigiDocService which provides SOAP-based web integration layer. To use this service, you need to have any kind of mobile phone and mobile network connection. The process of authentication for e-service requires 3 steps:

- Entering an e-service
- Requesting authentication using Mobile-ID
- Entering a PIN code on the phone

2.3.3 SplitKey based Smart-ID

SplitKey is an Authentication and Digital Signature Platform which uses end-user mobile devices (iOS and Android) to allow secure authentications. The solution can be used as an end-user access management tool for various e-services. SplitKey is fast and scalable solution that is secure based on threshold cryptosystem. The solution allows digitally signing documents according to the eIDAS directive of the European Union Regulation (Cybernetica AS, 2017).

The technology is a mixed token solution which combines the benefits of using a cryptographic hardware token and a software token. The main concept is to separate the private key into two parts and store them in separated systems. The concept consists of three main components. The first system is in the client mobile device and it is used for authentication and digital signing. The second one is the service provider on the server (also referred to as portal), which allows account management, user registration and customer support. The third is the core servers and HSM[4]'s. The important role of this server is to provide secure control of the client's private key. The server-side provides trust, auditability and assurance that a mobile device is not able to achieve independently. The client is the sole possessor of the full final key. Full final key can be used only by entering a PIN code from the device. Legally, the keys do not exist in isolation, they can exist only with client's personal identity certificates. There is a need to bind user's identity to onboard the newly created public key and results of SplitKey private keys (Cybernetica AS, 2018).

Smart-ID is main product that is powered by SplitKey technology. It is mostly used in Estonia and the Baltic states. The authentication and signature are recognized as highest level Qualified Electronic Signature in EU. In combination with recognition as QSCD (Qualified Signature Creation Device) Smart-ID is allowed to create signatures on the same level as ID-card or Mobile-ID based solutions. Smart-ID uses international scheme that is meant to be used across countries with different eID implementations. The solution already offers Smart-ID for identity providers, which means potential growth of the solution (SK ID Solutions AS, 2019).

To use this service, the user needs to have a smartphone with iOS or Android operating system and any kind of network connection. The process of authentication for e-service consists of 3 steps:

- Entering an e-service
- Requesting authentication using Smart-ID
- Entering a PIN code on the phone

2.3.4 Comparison

ID-card based solution needs the most steps to perform any action, because of the additional hardware it uses. Mobile-ID and Smart-ID solutions have the same number of steps. The difference between them lies in how the service is accessed and the type of network connection required. Mobile-ID is strongly mobile operator related and requires additional monthly fee to use it. The advantage of Mobile-ID from the usability perspective is only that it can be used by any mobile phone. The advantage of Smart-ID is that it can be used

[4] HSM - Hardware Security Module

on any iOS and Android device connected to any type of internet and it has no fees for the end user.

Integration for all of the services can be done using web service API. For ID-card and Mobile-ID solutions there are SOAP web services and Smart-ID uses a REST API. ID-card and Mobile-ID services are stricter to Estonian national eID PKI and are harder to implement in other countries, where Smart-ID has an advantage. Smart-ID offers an Identity Provider package for new eID providers which helps to expand the service usage.

2.4 Summary

The overview of wallet security solutions shows that none of them connects to user's identity which is needed for our solution. Having connection between identity of person and his actions will enable to protect participants' rights. This is one of the reasons why we start analyzing how to make it possible in our work. To support SQ4 Multisignature will be taken into account for further analysis in section 3.3.2. Multisignature allows to create own solution that can be integrated with any Ethereum Blockchain Dapp. Multisignature is code-based solution that does not require additional equipment to work. It will be an open-sourced code, that will make it possible to get feedback from users to improve the solution in the future.

In addition, we can state that Smart-ID is most accessible for user from other reviewed identity solutions. Smart-ID does not require additional equipment, only users mobile phone. It also does not have additional fees for end user and any internet connection can be used to access the service. The solution has an additional advantage because of REST API and possibility to integrate new identity providers. Smart-ID will be reviewed in more detail in section 3.3.3.

3 Specification of identity-based wallet integration with Dapp.

The creation of specification will consider explanation of the entities that will be used. Then in first part we will show what are the main requirements that we need to fulfill to solve the problem. We will use BPMN language to explain how current solutions, hereafter called AS-IS processes, work. These processes will be explained to see that current state is not fulfilling the requirements. In the second part we will extract information from existing solutions and analyze them. Analysis outcome will be used in solution specification, hereafter TO-BE process. The third part is modelling TO-BE process specification and its mappings to the requirements. BPMN models are including only information that is needed for scope of the specification.

The main entities of the model context are:

- Approver – the user who have rights to accept and sign transaction request;
- Asset – value stored in wallet;
- Authentication – process which confirms that user is exactly the person whom he calls himself;
- Dapp – web-based application that is connected to blockchain, which is used interact with smart contracts deployed in blockchain network and services available in the internet;
- Identity – user's state issued national identity;
- Smart Contract – program script code which is part of the system that runs on blockchain;
- Signature – value that is cryptographically signed using user personal private key;
- System – Combination of software in context;
- Transaction – identifier of asset transfer;
- User – person who has account on application or potentially creates one;
- User account – entry in application that stores user's information and allows user to use application;
- Wallet – Ethereum blockchain account with private and public keys.

3.1 Requirements

This section lists the requirements for the solution that are outlined by WePower Network[5] and refactored to fit the scope of this thesis. Functional requirements are listed in Table 3 and non-functional requirements for identity service are showed in Table 4.

[5] WePower Network - https://wepower.network/

Table 3 Functional requirements

ID	Requirement
R1	User should sign up to the Dapp only with identity authentication.
R2	User should have an asset wallet that is connected with identity.
R3	User should transfer assets to another user only after signing transaction with identity signature.
R4	Approver should approve assets transfer.
R5	Approver should approve transfer by signing transaction with an identity signature.

Table 4 Non-functional requirements

ID	Non-Functional Requirement
NFR1	Identity service should not require additional fees from the user.
NFR2	Actions with identity service should not require additional pluggable equipment.

3.2 AS-IS processes specification

This section describes AS-IS process models that are related to the requirements. Aim is to get overview of the processes and target tasks that should be updated with required functionality.

Figure 4 shows main tasks that are followed to sign up to application. The process includes two participants user and application. Model shows that blockchain wallet is created separately of the application and wallet protection solution (one of mentioned in chapter 2.2) is used to protect the wallet private key. User signs up to application and uses wallet address to link his account to the wallet.

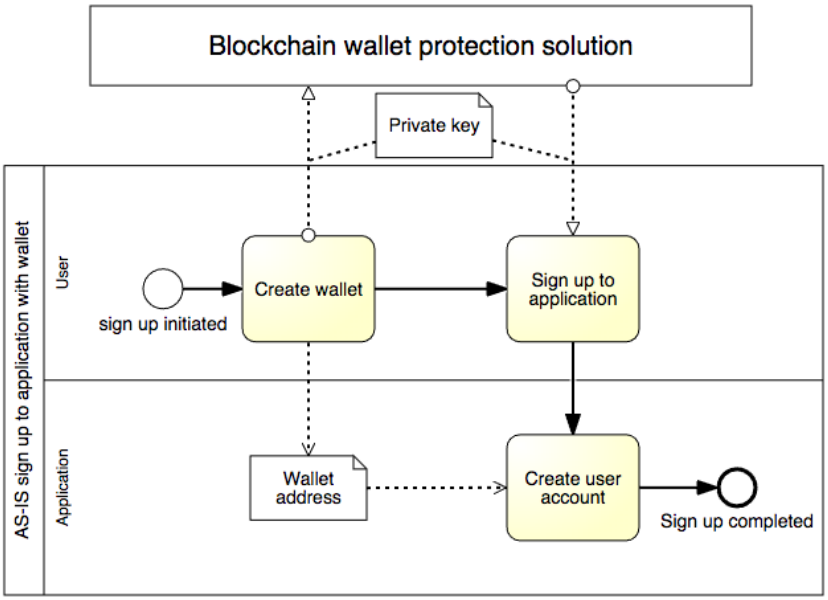

Figure 4 AS-IS sign up

Figure 5 describes process of making transaction. The model shows that after user have successfully logged in, the initiation of transaction should be triggered and after that application prepares the transaction and asks private key. Private key is accessed by user using one of wallet protection solutions (mentioned in chapter 2.2) and provided to application to perform transaction.

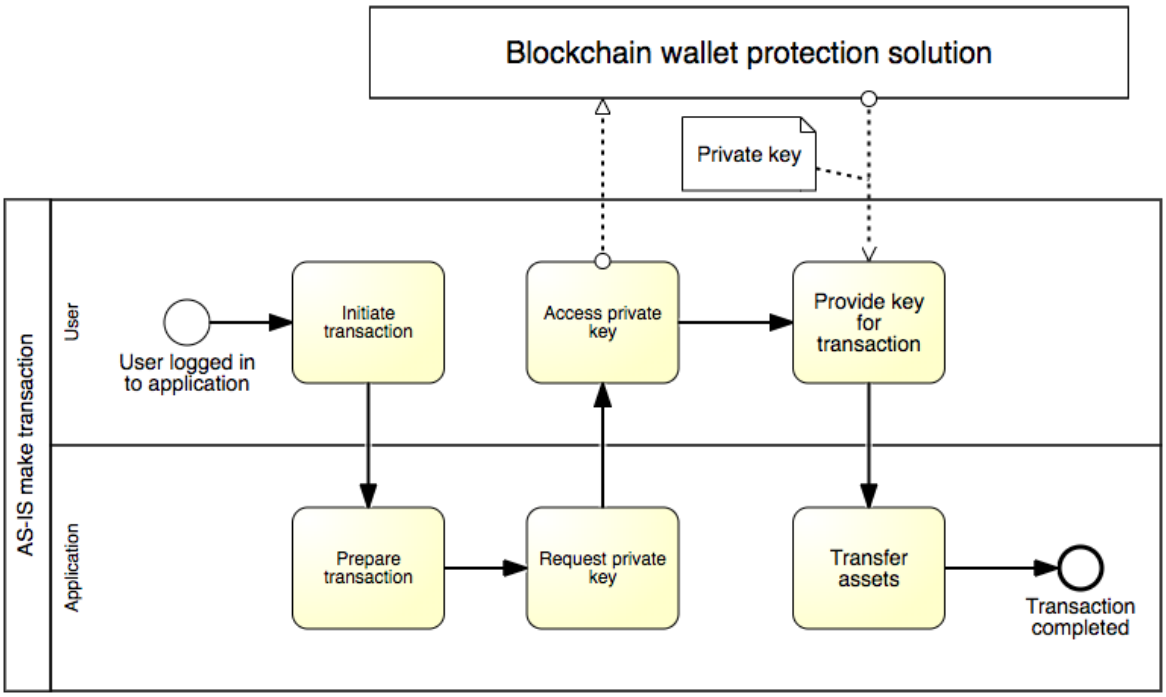

Figure 5 AS-IS make transaction

On the models we can see that user is not connecting his identity to the application. The transaction model shows that there is no approval step and transaction is not connected to users identity signature. Overview of AS-IS processes relation to requirements are concluded in Table 5. This table shows that none of the requirements are fulfilled in AS-IS sign up and transaction process.

Table 5. AS-IS processes relation to requirements. Requirement not fulfilled is indicated as " - " and not related marked as empty cell.

Process	R1	R2	R3	R4	R5	NFR1	NFR2
Sign up	-	-					
Make transaction			-	-	-		

3.3 Analysis of existing solutions

To create TO-BE specification for the solution there will be three different aspects analyzed. The first part shows existing example of identity-based signing in smart contract. Analysis of Multisignature examples is explained in the section 3.3.2 and processes of Smart-ID in the third section 3.3.3.

3.3.1 Identity service integration with smart contract - Ethstonia example

This chapter analyzes how Ethstonia solution works. Ethstonia was a prototype which is connecting government issued ID-card to Ethereum blockchain wallet and allow to sign transactions using this ID. It is in prototype stage, so there was no interface online. The solution is web page application, which connects to the Ethereum blockchain. Application needs to be run from source code, and it needs ID- card from Estonia to apply the transaction. There were no specification how the wallet itself was created and where the actual keys of Ethereum wallet were stored (Logvinov, Ethstonia Identity, 2018).

To investigate the solution source code (Logvinov, 2019), the prototype was reviewed and executed. The outcome of the review is visualized using BPMN model. Figure 6 shows the main process flow of making transaction with three actors – user, application (Dapp) and Smart contract. Tasks explain the flow of transaction process and data object usages. This solution creates signature on hashed object that contains receiver wallet address and value to be transferred. The same object and its signed values are sent to smart contract to perform signature verification. After successful signature verification transaction is sent to receiver with specified value.

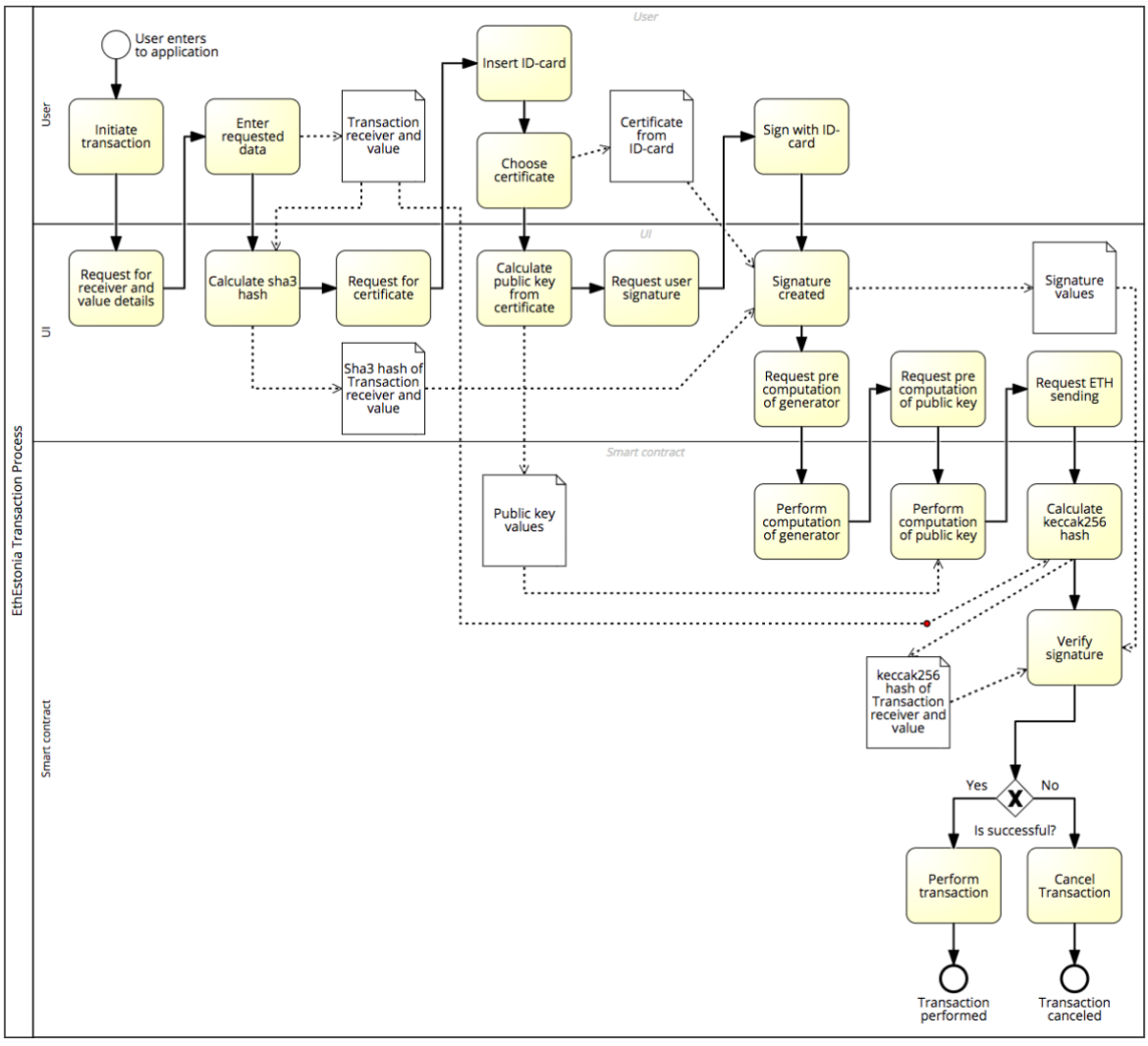

Figure 6 Ethstonia transaction process

This solution is created using Truffle framework and the same framework that provides local network is used for execution. Execution of this solution is limited and cannot be fully performed due to costs of the computational tasks and signature verification task. The Gas values used in transactions of the solution exceeds Gas limit values that are currently used in live network, which is ~8M of Gas per block (Etherscan, 2019).

Outputs of Ethstonia example analysis is listed in Table 6. Ethstonia example shows that signature must be verified and verification process is too expensive process in blockchain network. As conclusion we see that it is important to have sufficient data to be signed with identity signature and the calculations should be optimized to be cost efficient.

Table 6 Extracted information from Ethstonia example.

Specification topic	**Extracted information**
Data to be signed with identity	There should be necessary data to be signed. In case of Ethstonia there is only receiver and value, which is not enough to associate specific transaction with user's signature.
Hashing algorithm	Ethereum calculates hashes with SHA3/keccak256.
Signature verification	Every signature created with the identity should be verified before executing the transaction.
Calculations costs	Calculations for signature verification might require more Gas than can be used in the network. Calculations performed in the smart contract should be simple to be executable on the chain
Solidity Framework	Truffle Framework to write smart contracts and Dapp

3.3.2 Multisignature wallet contracts

This section describes results of the analysis of three different Multisignature smart contract solutions. The main feature of Multisignature contract is make blockchain transactions by acceptance of multiple users. This is exactly what is needed to fulfill requirement R4 where one user will be who wants to perform transaction and another one who approves it. Purpose of this analysis is extracting knowhows and finding suitable examples for solution to be implemented.

According to overview of Multisignature contracts (Unchained Capital, 2018), there are some contracts that lost their trust due to code vulnerability issues. ConsenSys, Gnosis and BitGo Multisignature Wallet contracts are still trustful and used in production. Trust of the Multisignature contract is important and implementing a solution from scratch is not reasonable. Source codes of contracts are reviewed to find reusable features of existing contracts. The output of the reviews is additional input for decisions made for solution's TO-BE specification.

ConsenSys Multisignature Wallet (George, 2017) and Gnosis Multisignature Wallet (George, 2019) source codes are from the same author and the functionality of the contracts are nearly the same. Common features and specifics of the contracts are as follows:

- Constructor defines an array of owners' addresses and a number of required confirmations for asset transaction (up to 50).

- Allows to add, delete, replace owner addresses. Method should be executed by the wallet address itself.
- Allows to change required number of confirmations by walled address.
- Separated methods for transaction submission and confirmation.
- Submitted transaction is stored as a struct that holds all the required data to execute the asset transfer when needed.
- No transaction signatures used, submitted transaction gets its id that other owners should confirm.
- Owners can revoke their confirmation.
- Transaction can be executed by anyone, if required number of confirmations condition are filled.

In addition to common features, ConsenSys contract has daily withdraw limit and Gnosis has additional features to call current state of the transactions and confirmations.

BitGo Multisignature Wallet v2 (Allmendinger, Chan, & barathcj, 2017) source code is reviewed, and main features and specifics are extracted:

- Constructor defines exactly 3 owners' addresses.
- Uses modifier (requirement check) for all public methods.
- Uses sequence in each transaction to prevent replay attacks and verify if the same sequence transaction has not been called before.
- Transfer assets requires signature argument, which is created using keccak256 with arguments prefix, receiver address, value, data, expire time.
- Uses one method to perform asset transaction to new address, first owner is taken as sender address and second from signature of one of the other participants (two participants of three in total).
- During transaction method execution operation hash is created to retrieve second participant from signature hash.
- Uses additional feature called safe mode to prevent asset transferring to wrong addresses.

Main difference of these contracts is that how many methods should be called before the execution, due to that BitGo contract needs to create its own signature that is verified before the transfer. Outcome of the review has a number of features that are listed in Table 7. According to requirements R3, R4, R5 there is a need to have multiple steps before execution, because first user needs to initiate the transaction and this transaction should exist before approval. In that case solution should use combination of reviewed contracts. To conclude the table, we can say that most of the basic methods and features will be reused in solution implementation.

Table 7 Extracted information from Multisignature wallet analysis.

Source	Specification topic	Extracted information
Gnosis, ConsenSys	Transaction	Create transaction struct to store all transactions and make them accessible by id. Use boolean "executed" for transaction state tracking. Use different methods for transaction initiation and confirming.

BitGo	Hashing algorithm	Keccak256 (Sha3) algorithm used to hash used information in smart contract.
Gnosis, ConsenSys, BitGo	Modifier	Modifier used to check conditions before method execution in smart contract.
Gnosis, ConsenSys, BitGo	Address	Store addresses of contract owners during contract creation.

3.3.3 Smart-ID signing and authentication process

The section describes the process of Smart-ID based authentication and signing. Main entities, execution steps and specifications are extracted and summarized. Output shows specifications identity related functionality needed to fulfill the requirements.

Entities of Smart-ID process are as follows:

- Relying Party (RP) – The service that initiates Smart-ID processes, in case of this project the Dapp.
- Certificate – certificate that belongs to certificate authority, which Public Key values (exponent and modulus) are needed to verify signature.
- Data – hashed value of the document to be signed or random generated value in case of authentication
- Certificate Authority (CA) – Authority that holds certificate of the person's identity
- Verification code – The code that shows up on RP and mobile device which indicates that signing or authentication is performed on the same data on both sides.
- Signature – object of value and algorithm that is result of signing or authentication process. It can be used to verify the validity of signed data.
- Mobile device – device receives authentication or signing requests. Device where user confirms actions by entering a PIN code.
- Smart-ID backend – application of Smart-ID system that interacts with RP, Mobile device and CA.
- Person's identifier – identifier of person's identity (example in Estonia 38012121212)

Smart-ID uses REST API based interface which allows the execution of signing and authentication processes. As our solution needs to integrate with Smart-ID we will show more detailed process visualization. Figure 7 is a sequence diagram that describes detailed authentication and signing process. The flow starts with request from Relying Party with person's identifier and random data hash, and which receives the session id. Smart ID backend performs request from CA and starts user authentication on a mobile device. Relying party is polling for response of session until user confirmation on mobile device. Once user confirms authentication end result is returned (Jalukse, 2019).

Signing process consists of same steps as the authentication process, except that it has no interaction with CA. Second difference is that the data object of first POST request includes document hash that signature is requested for.

The OSCP[6] service can be used separately for checking certificate validity after the signature is made. This is the feature that supports SQ6, as we can always validate user's identity to have a proof of valid signature.

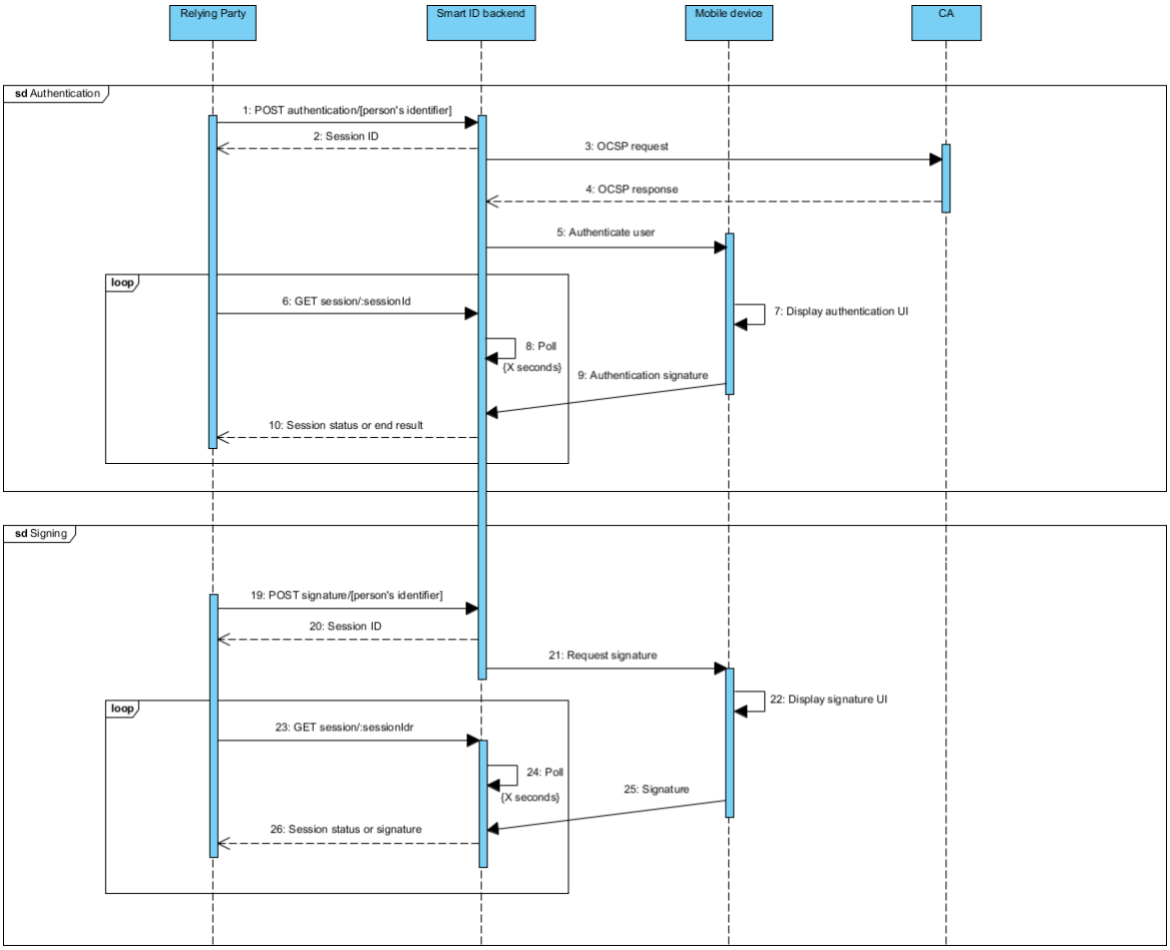

Figure 7 Smart-ID authentication and signing process sequence diagram (Nõmmik, 2017)

Conclusion of Smart-ID analysis is shown in Table 8. For authentication or signing with Smart-ID the data should be hashed using SHA-2 type algorithm. Verification of signature is performed using RSASSA-PKCS1-V1_5 standard (Internet Engineering Task Force, (IETF), 2016), which is a mathematical calculation with 768-byte sized number. Execution of sample authentication and signing requests gives different certificate results. It means that signature cannot be verified with authentication certificate. As authentication and signing is in theory the same process, signing request can be used as authentication. To verify that user receives correct request of authentication or signing, verification code calculation is performed using data hash and output consisting of 4 digits is displayed on the mobile device and the service. Hashing algorithm, signature verification and the steps through Smart-ID flow will be used in implementation of solution prototype.

[6] OSCP (Online Certificate Status Protocol) – service of the warranty of user certificate validity.

Table 8 Extracted specification from Smart-ID

Specification topic	Extracted information
Hashing algorithm	Supports only SHA-2 types of data hashing.
Signature	Verification standard RSASSA-PKCS1-V1_5.
Certificate	Signature certificate should be stored for verification.
Signing and authentication process	Requires two requests to get data (initiation and getting session result).
Verification code	Is calculated as: integer(SHA256(data_hash)[−2:−1]) mod 10000

3.4 TO-BE processes specification

This section proposes TO-BE process models that fulfill the requirements. AS-IS processes are used as a starting point and specifications extracted from analyzes discussed in section 3.3 are applied. All processes are modelled as success flow of the solution. As success flow we consider going through the user interaction process without any errors.

AS-IS sign up process is creating connection between user and Dapp using only wallet address. The TO-BE model aim is to connect specific user wallet which is connected to user identity-based signature certificate provided by Smart-ID service.

Figure 8 describes the sign up process to Dapp. Smart-ID is showed as part of the process to describe how R1 is fulfilled. Dapp requests user authentication and received certificate is passed to the new smart contract along with approver certificate, approver address, user personal id and wallet address. The new smart contract is instance of a new Identity-Based Multisignature Wallet, hereafter called IBMSW in this paper. User wallet address is used to interact with their IBMSW. Process results with new IBMSW deployed to blockchain network and new account is created in Dapp, which fulfills requirement R2.

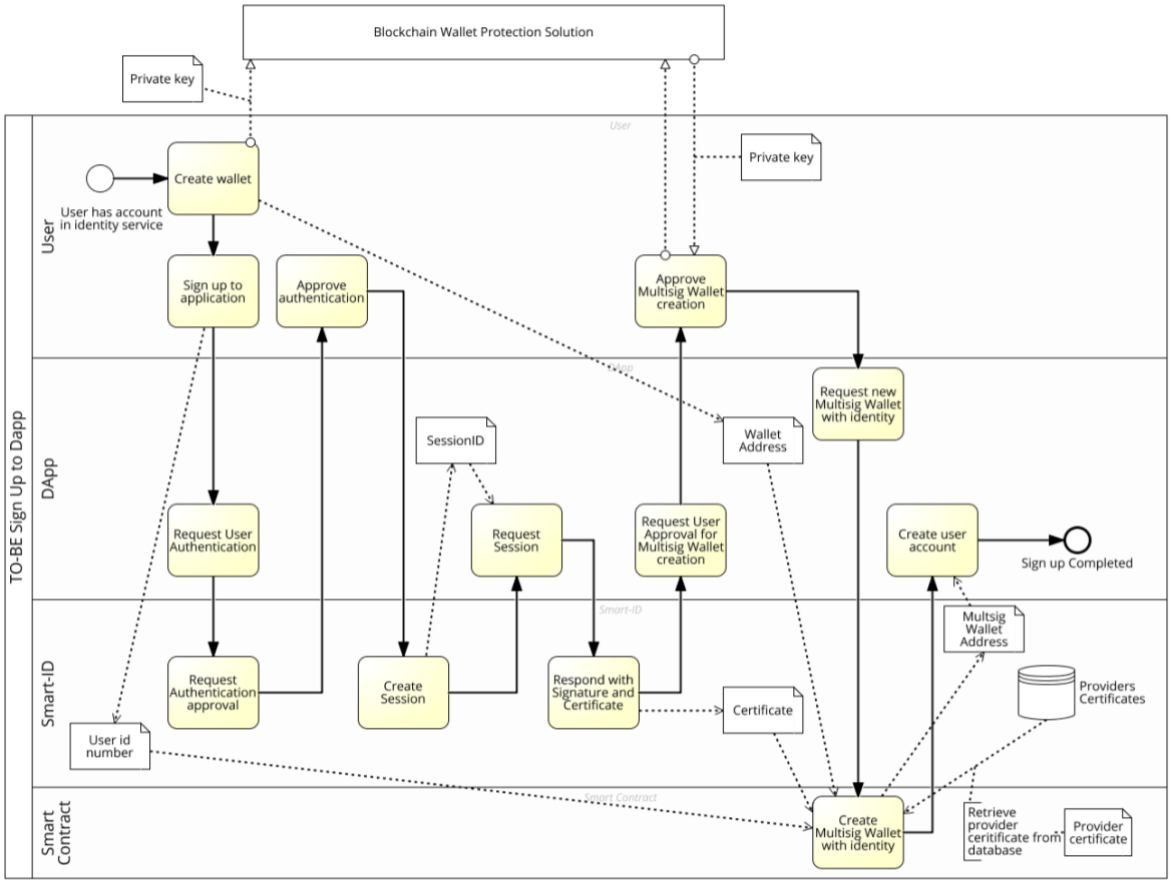

Figure 8 TO-BE sign up to Dapp

As user will have personal IBMSW after TO-BE signup process the AS-IS transaction process is extended with interactions between the contract and the Smart-ID service.

Figure 9 describes transaction process using IBMSW. We are not mentioning the private key usage from Blockchain Wallet Protection Solution in the following models to be clearer on the interaction between Smart-ID, User, Dapp and Smart contract. We mention that the private key is used for every interaction with Smart Contract.

A transaction process can be initiated only by user of Dapp. After user initiates and enters receiver address and value, a new record of transaction is created in IBMSW. Next step is to sign the transaction object that is created during the initiation. Signing process is performed with the hashed value of the transaction object. After signing is confirmed by the user, the signature is verified in IBMSW and added to transaction record. This step fulfills requirement R3, as it cannot be skipped. Next task is subprocess of transaction request approval, described in Figure 10. It is same the signing and verifying process as it is for the user, except that signing is performed by approver. By completing approval process, approver signature is added to transaction record. This shows that requirement R4 and R5 are fulfilled, as transaction process cannot go further without this step. Last step of transaction is execution by the user, where confirmations are checked and value is transferred to the receiver.

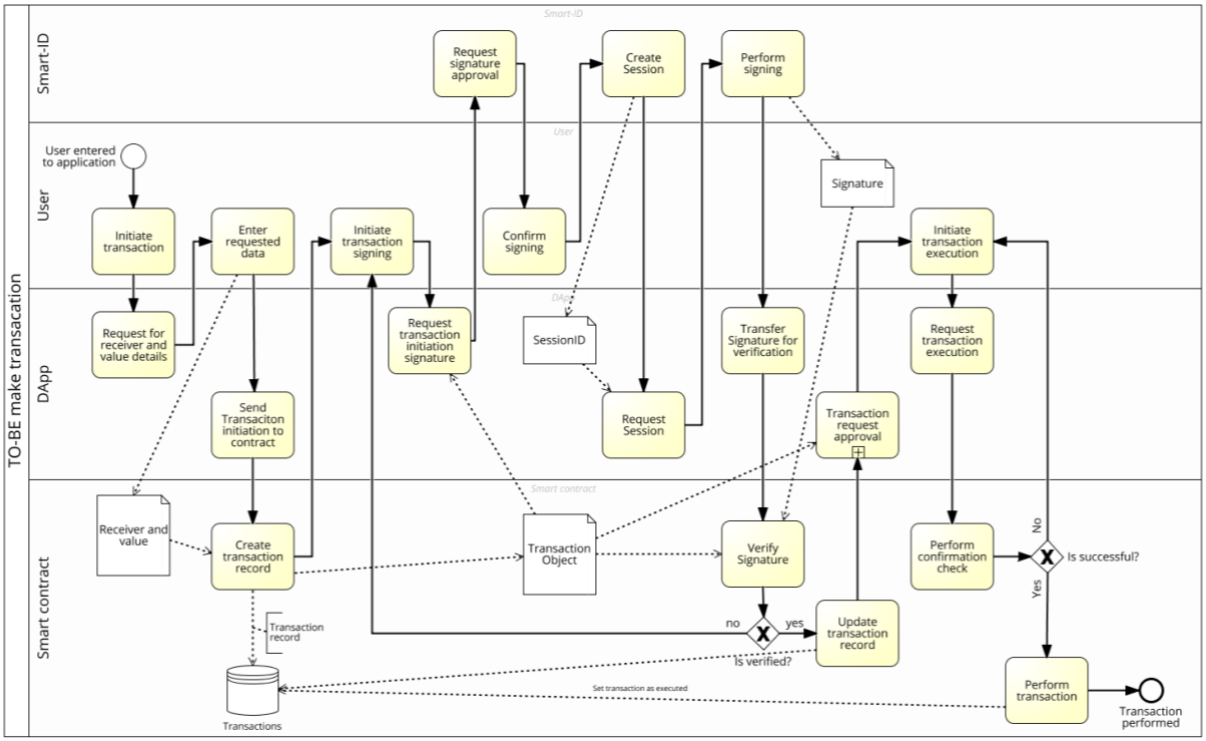

Figure 9 TO-BE make transaction

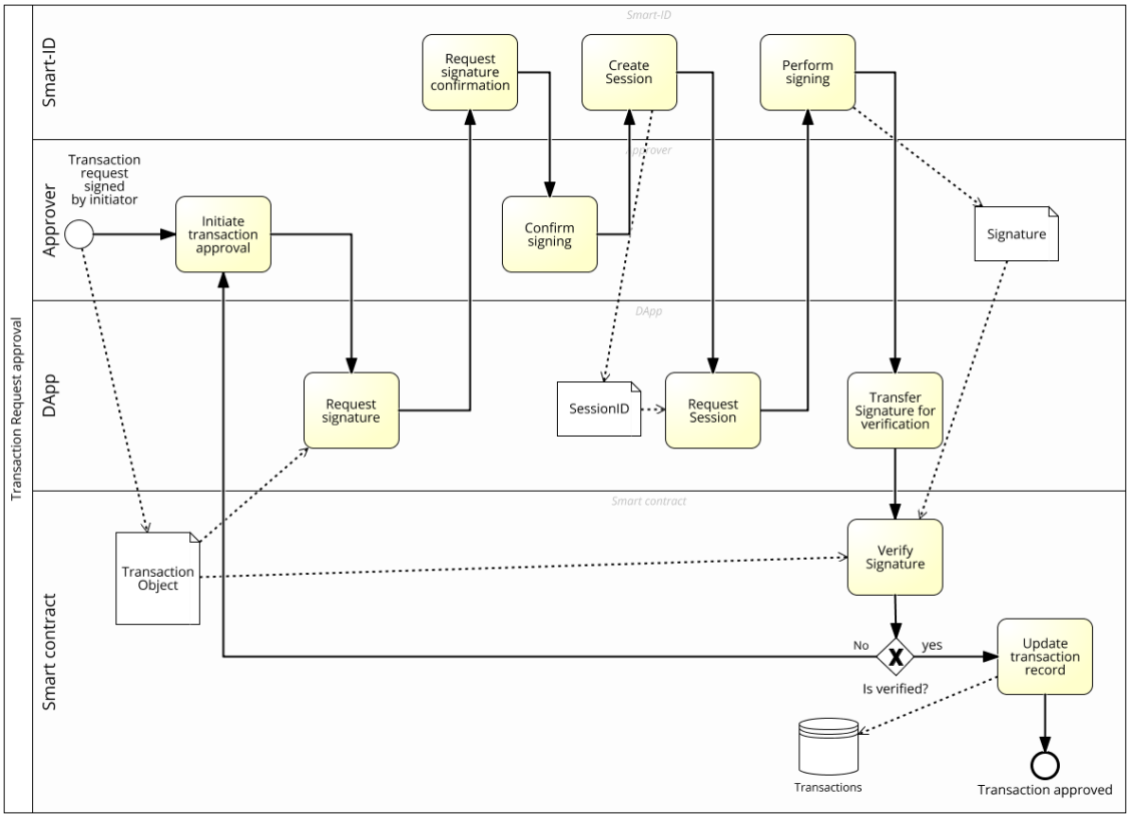

Figure 10 Transaction approval subprocess

Overview of requirements and relation with TO-BE processes is concluded in Table 9**.** Non-functional requirements are marked fulfilled as Smart-ID uses smartphone application for authentication and signing and Smart-ID is free of charge for end user.

Table 9 TO-BE processes relation to requirements. Requirement fulfilled is indicated as " + " and not related marked as empty cell.

Process	**R1**	**R2**	**R3**	**R4**	**R5**	**NFR1**	**NFR2**
Sign up	+	+				+	+
Make transaction (with approval subprocess)			+	+	+	+	+

In summary the main concept of TO-BE models is to integrate identity based Multisignature wallet into the Ethereum Dapp, also named as IBMSW. The Blockchain Wallet Protection solution that holds Ethereum wallet private key, will still remain to be free of choice for the user. Multisignature will only be additional layer in the process. This layer is connecting users' identity to blockchain to enable signing the transaction with the identity signature through the Smart-ID. This enables us to fulfill all of the requirements raised in section 3.1.

4 Validating specification with prototype

This chapter describes the implementation of the prototype that validates TO-BE process models discussed in section 0. Output of the existing solutions analysis from section 3.3 will be used as inputs. Prototype includes smart contract and Dapp implementation.

4.1 Main frameworks and architecture

Truffle framework has been chosen for smart contract implementation and testing in local Ethereum blockchain network. Truffle is using Ethereum Virtual Machine (EVM). Framework provides support for writing, debugging, migrating and deploying contracts written in solidity. Truffle has development boilerplates that helps to start implementing exactly what is needed without setup configuration.

Dapp is created using JavaScript ReactJS framework. This framework is one of the most popular user interface frameworks in JavaScript. It has largest recommendations percent according to 2018 reports (Greif, Benitte, & Rambeau, 2018). Connection to Ethereum network from ReactJS was provided using web3.js library (Ethereum, 2019). Smart-ID integration was implemented using HTTP REST API requests to Smart-ID demo instance (SK-EID, 2019).

General software architecture of the prototype is presented in Figure 11.

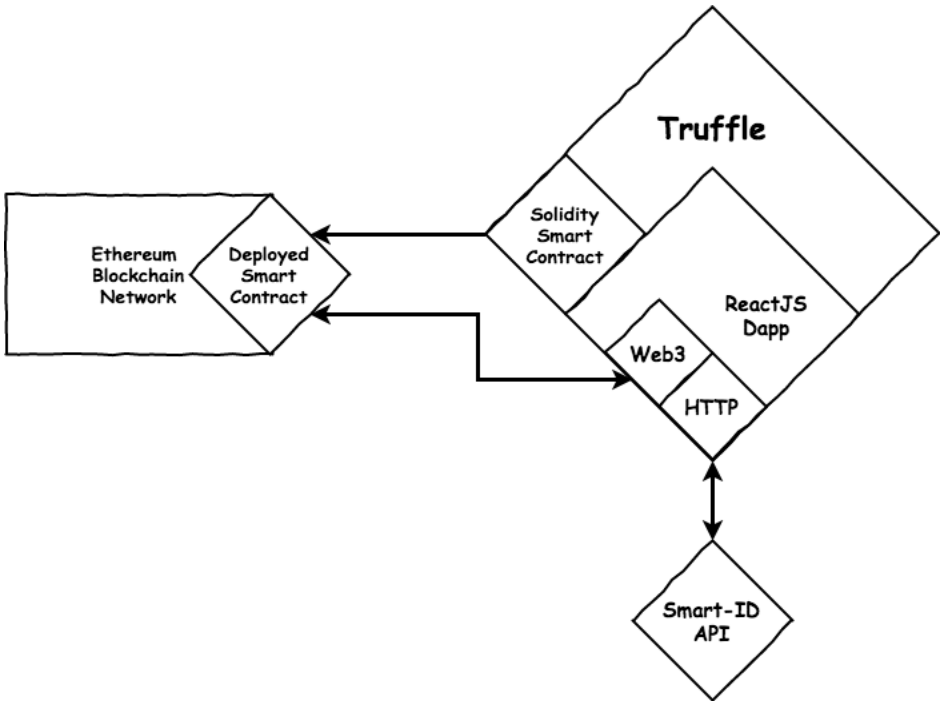

Figure 11 Architecture of prototype software

4.2 Validation example

The prototype consists of two parts: the Dapp and the smart contract. The main logic is written in smart contract part and Dapp is used for interaction between Smart-ID and smart contract.

To validate TO-BE specification we will go through sign up, transaction and transaction approval processes using prototype.

To sign up user needs to provide their certificate that is given by Smart-ID service. First step is to enter user identity code in Dapp and signing a document. For prototype we used secure random generated hash (generated the same way as Smart-ID does authentication hash). After user signs the hash on their smartphone Dapp receives user's certificate. With user certificate provided we use Dapp to create an instance of smart contract in Ethereum network. The user certificate is stored as owner certificate, that will be used to verify further signatures. In addition to user's we save approver's certificate. Which should be securely provided from Dapp in the real case, but to simplify our prototype, we use the same user's certificate from another device. During this process we created new wallet that is connected to user's identity with respect to requirements R1, R2, NFR1 and NFR2.

To make a transaction user first needs to create a request in their wallet contract with amount and receiver address specified. Each request creates new record in smart contract. The data of transaction request is hashed and passed to Smart-ID service for signing. User signs hashed data using their smartphone and signed data is validated in smart contract using user's certificate. The signature is stored in request record. The next step is to sign the data by approver, which certificate was written to the smart contract on creation. Approver signs the data same way using Smart-ID service. Their signature is validated using approver's certificate and stored in request record. Transaction can be executed when both validated signatures are stored in transaction request record. Execution results with asset transaction to specified receiver address and amount. This process fulfills requirements R3, R4, R5, NFR1 and NFR2.

As addition, the request record is accessible by its id to check its status. There can be multiple request created in parallel and executed in different order. This process works successfully following the TO-BE specification.

4.3 Prototype project structure

The prototype is an open-source project, which structure will be described in this section. A Truffle project is used as root which contains main truffle folders - *contracts, migrations, test* and *truffle-config.js* as shown in Figure 12 on the left. Solidity code is stored in *contracts* folder. Migration scripts specify Solidity contracts that will be deployed to network during truffle migration. Truffle network configuration and compilation output folder is configured in *truffle-config.js*. Compilation output is set directly to *dapp/src/contracts* folder, because Dapp is using compiled version of contracts.

Dapp is created under truffle project in folder *dapp* shown in Figure 12 on the right. *Dapp.js* is main application script that combines all libraries, utilities and defines user interface. Scripts that helps to perform communication with Smart-ID (*smartIdUtil.js*), parse

certificates and signatures (*cryptoUtil.js*) and create connection to Ethereum network (*getWeb3.js*) are located in *utils* folder.

Most of the project can be publicly found on Github (https://github.com/aleksandertsg/IBMSWallet)

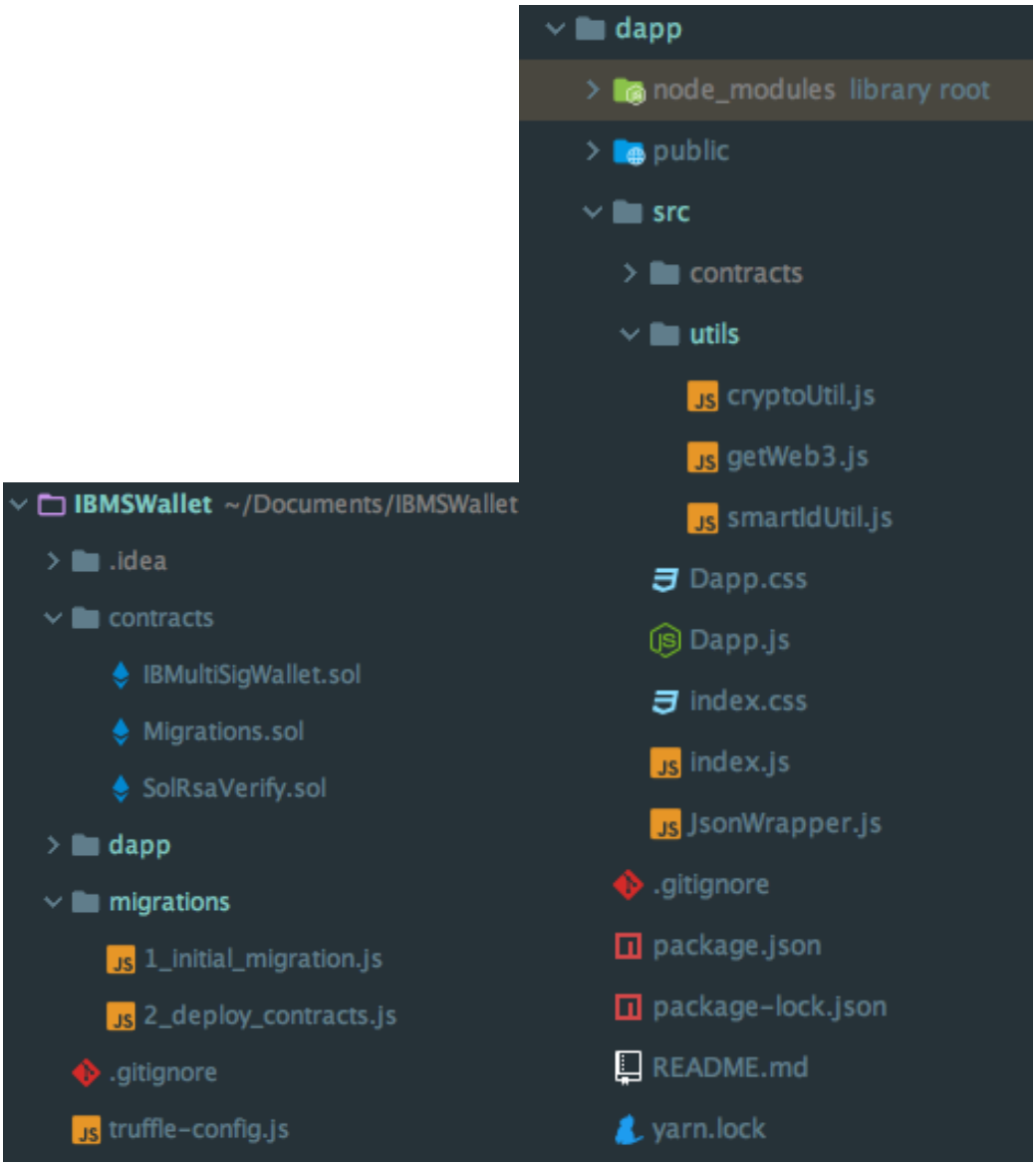

Figure 12 Prototype project structure

4.4 Main functionality

In this section we will have an overview of smart contract, utilities and Dapp. These are the core components that interacts with each other according to the specification in section 0. The following sections will show fragments of the code and its functionality.

4.4.1 Smart contract

Main functionality of *IBMultiSigWallet.sol* file (other functions are not shown):

```
contract IBMultiSigWallet {
  ...
  constructor (bytes memory oModulus, bytes memory oExponent, uint id, address
_approver, bytes memory aModulus, bytes memory aExponent) {...}

  function initiateTransfer(address payable receiver, uint256 value) {...}
  function signTransaction(uint transactionId, bytes memory signature, bytes32
messageHash, bytes memory message) {...}
  function approveTransaction(uint transactionId, bytes memory signature) {...}
  function executeTransaction(uint transactionId) {...}
  function verify(bytes32 msg_hash, bytes memory signature, bytes memory m,
bytes memory e) {...}
  ...
}
```

Function *constructor(…)* is used to create new wallet. It takes user certificate modulus, exponent, personal id, approver address, approver certificate modulus and exponent. All of the arguments are stored in wallet and these values cannot be changed.

Functions *initiateTransfer(…), signTransaction(…)*, approveTransaction(…) and executeTransaction(…) are transaction process methods that represents smart contract tasks described in chapter 0. These functions can be executed only in order they are listed. And every call is checked for allowance.

Function *verify(…)* is verifying signature against message hash using RSASSA-PKCS1-V1_5 standard. This function uses external *SolRsaVerify.sol* contract to make the calculation (Massanet, 2018).

4.4.2 Utilities

The main functionality of the utilities is to handle cryptographical calculations and perform data transferring and processing in Dapp.

File *smartIdUtil.js* contains function:

```
request = async (type, docID, hash, hashType = 'SHA256', displayText) => {...}
```

Function *request(…)* makes HTTP request to Smart-ID and waits for signature response. It performs two requests to get the value, as described in section 3.3.3.

File *cryptoUtil.js* has three functions that are needed to perform cryptographic calculations for signature and data:

```
getPublicKey = (cert) => {...}

getDataHash = (hashType = 'sha512', data = secureRandom(64)) => {...}

calcVerifyCode = async (buf) => {...}
```

Function *getPublicKey(…)* is parsing exponent and modulus values out of given certificate hash. These values are passed to contract while user registers new IBMSW contract.

Function *getDataHash(…)* calculates base64 value of data after hashing with specified hashType.

Function calcVerifyCode(…) used data to calculate verification code that is displayed on Smartphone and Dapp while authentication or signing operation from Smart-ID is requested.

4.4.3 Dapp prototype

In Figure 13 we can see Dapp user interface application that helps to interact with IBMSW contract and Smart-ID. Using that interface, we can validate sign up and transaction TO-BE processes from chapter 0.

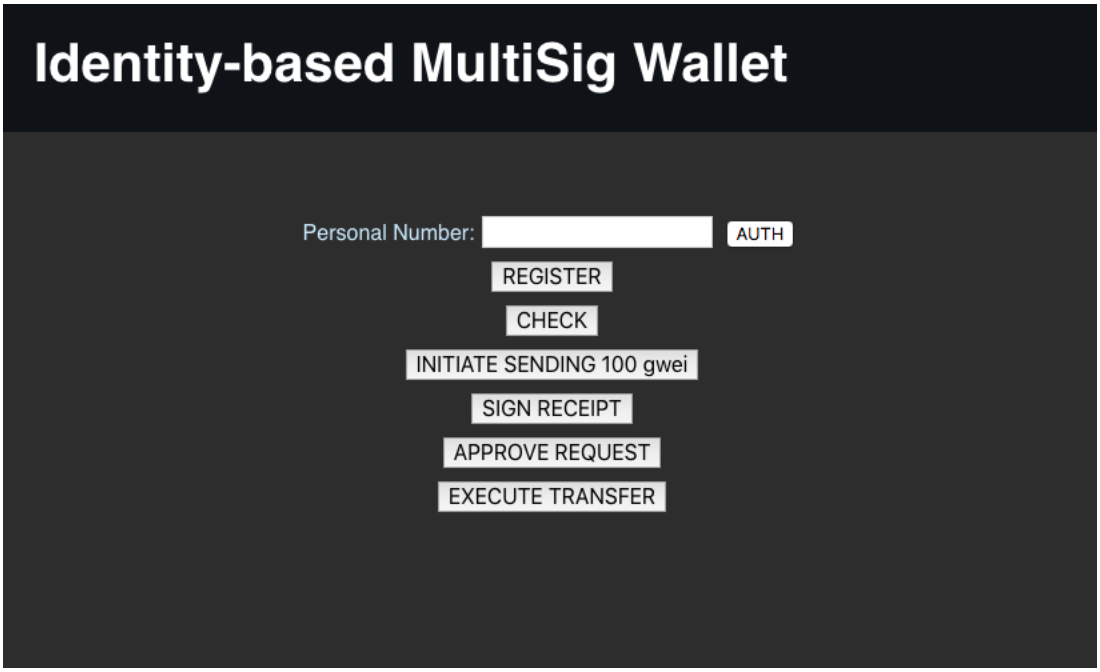

Figure 13 Dapp prototype user interface

5 Conclusion

In this work we created an overview of the Ethereum blockchain entities and working principles. We analyzed different wallet protection solutions and user identity verification and signature services and selected most suitable ones. The requirements for solution were listed and checked against current state. We pointed out that none of the requirements were fulfilled and continued analyzing existing solutions on more deep level. The outcome of this analysis was used to create the solution specification that fulfills all of the requirements. To validate the solution specification, we developed a prototype that includes Multisignature contract, simple Dapp and additional utilities. As a result, we have a solution specification which is successfully validated with the prototype. After the completed work we can answer for all of the research questions:

How to connect the user identity to the wallet in a way that transactions can be made only after identity-based signature verification? We created a Multisignature contract that holds user's certificates from Smart-ID service. The transaction is signed using Smart-ID and certificates are used for transaction validation before execution.

SQ1 How to make decentralized application know who is making the transactions? The transaction cannot be performed without identifying the user, which is connected to Smart-ID service through decentralized application.

SQ2 How to associate an asset transfer with user identity? In our solution, smart contract stores a record of transaction with signature that is validated using user certificate given by Smart-ID service.

SQ3 How to integrate state-level identity signing service into the Ethereum blockchain? We used a user certificate, that is stored in smart contract and integration with Smart-ID service in decentralized application.

SQ4 How to create a wallet with multiple transaction approval steps? The Multisignature wallet is stores two certificates on creation one for the user and one for the approver. No transaction can be performed without approver signature.

SQ5 How to verify identity-based signature in Ethereum blockchain? Each signed transaction value is unpacked in smart contract using user's certificate. The equality of extracted value and unsigned transaction value is compared.

SQ6 How to check that transaction made in the past, was made with valid identity? Smart-ID provides an OSCP service that allows to check validity of the certificates at any time.

The limitations of this work were added in the beginning and during the progress:

- There were no academic papers about national identity connection to blockchain signing process found during the research. Source code of examples and non-academic papers were used as references.
- Solution is limited to be used on Ethereum Blockchain, because the smart contract is written in solidity language. To use the solution in other Blockchains there is a need to rewrite the code in another language.
- Identity services considered in the solution are national identity verification and signing services of Estonia.

- To enable more countries to use the solution additional integration is needed. As limitation of this thesis, user wallet private key security was not taken into account. Main focus was to create solution model considering requirements and prototype of smart contract and Dapp.
- In prototype we used simply hard coded approver certificates, because the focus is to make it possible to work. Providing approver certificates

The outcome of the thesis answers all of the research questions and fulfills the requirements, however we highlight some points for future work:

- Add possibility to have multiple approvers.
- Implement service that retrieves approver certificates on new contract creation.
- Create possibility for user to use multiple certificates.
- Implement error handling for execution failures.
- Create tests for smart contract

References

Al-Jaroodi, J., & Mohamed, N. (2019, 03 07). Blockchain in Industries: A Survey. *IEEEE Acces, 7*, pp. 36500-36515.

Allmendinger, O., Chan, B., & barathcj. (2017, 11 23). *eth-multisig-v2/contracts/WalletSimple.sol.* Retrieved from Gtihub: https://github.com/BitGo/eth-multisig-v2/blob/master/contracts/WalletSimple.sol

Andreas M. Antonopoulos, G. W. (2018). Mastering Ethereum: Building Smart Contracts and DApps. In *Andreas M. Antonopoulos, Gavin Wood Ph.D.* (pp. 59-160). Sebastopol: O'Reilly Media. Retrieved from https://books.google.ee/books?id=nJJ5DwAAQBAJ&pg=PA60&dq=private+key+in+ethereum&hl=et&sa=X&ved=0ahUKEwiAnNOHipjfAhXGECwKHXh5CUQQ6AEIJzAA#v=onepage&q=private%20key%20in%20ethereum&f=false

Bamert T., D. C. (2014). BlueWallet: The Secure Bitcoin Wallet. Switzerland: Springer, Cham.

Berryhill, J., Bourgery, T., & Hanson, A. (2018). *Blockchains Unchained*. (OECD Working papers on Public Governance, no. 28, OECD Publishing, Paris.) Retrieved 12 12, 2018, from https://read.oecd-ilibrary.org/governance/blockchains-unchained_3c32c429-en#page1

Blockgeeks. (2018, 09 13). *What is Blockchain Technology? A Step-by-Step Guide For Beginners*. Retrieved 12 12, 2018, from https://blockgeeks.com/guides/what-is-blockchain-technology/

Bruno. (2017, 09 08). *How Do Hardware Wallets like the Ledger Nano S Work?* (Bitfalls) Retrieved 12 15, 2018, from https://bitfalls.com/2017/09/08/hardware-wallets-like-ledger-nano-s-work/

Cheetah Mobile Blockchain Research Lab & Cheetah Lab. (2018). *2018 Global Cryptocurrency Wallet Security White Paper*. Retrieved 12 14, 2018, from https://www.cmcmbc.com/protocol/safe-wallet/white-paper/

Christian Sillaber, B. W. (2017, 08). Life Cycle of Smart Contracts in Blockchain Ecosystems. *Datenschutz und Datensicherheit - DuD, 41*(8), 497-500. Retrieved from https://link-springer-com.ezproxy.utlib.ut.ee/content/pdf/10.1007%2Fs11623-017-0819-7.pdf

Christidis, K. (2016). *Blockchains and Smart Contracts for the Internet of Things*. (IEEE Access) Retrieved 12 12, 2018, from https://ieeexplore-ieee-org.ezproxy.utlib.ut.ee/stamp/stamp.jsp?tp=&arnumber=7467408&tag=1

Coinbase. (2019). *Secure Bitcoin Storage*. Retrieved 04 02, 2019, from https://www.coinbase.com/security

Cybernetica AS. (2017). *SplitKey Authentication and Digital Signature Platform.* Retrieved 12 16, 2018, from https://cyber.ee/products/digital-identity/materials/splitkey-brochure.pdf

Cybernetica AS. (2018, 12 12). *Introduction to SplitKey Foundations White paper.* Retrieved 12 16, 2018, from https://cyber.ee/products/digital-identity/materials/splitkey-whitepaper.pdf

Destefanis, G., Marchesi, M., & Ortu, M. (2018). *Smart Contracts Vulnerabilities: A Call for Blockchain Software Engineering?* (IEEE) Retrieved 12 10, 2018, from https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8327567&tag=1

e-estonia. (2019). Retrieved March 2019, from https://e-estonia.com/solutions/e-identity/

Estonia, I. S. (2019, 04 11). *ID*. Retrieved 04 11, 2019, from https://www.id.ee/

Ethereum. (2019, 03 23). *Introduction to Smart Contracts.* Retrieved from Github: https://github.com/ethereum/solidity/blob/develop/docs/introduction-to-smart-contracts.rst

Ethereum. (2019, May 2). *web3.js - Ethereum JavaScript API*. Retrieved from Github: https://github.com/ethereum/web3.js

Etherscan. (2019, 05 2). *Blocks*. Retrieved from Etherscan: https://etherscan.io/blocks

Fu, Y. (2017, 09 12). *Off-Chain Computation Solutions for Ethereum Developers.* Retrieved from Medium: https://medium.com/@YondonFu/off-chain-computation-solutions-for-ethereum-developers-507b23355b17

George, S. (2017, 10 6). *ConsenSys/MultiSigWallet/MultiSigWalletWithDailyLimit.sol.* Retrieved from Github: https://github.com/ConsenSys/MultiSigWallet/blob/master/MultiSigWalletWithDailyLimit.sol

George, S. (2019, 02 18). *gnosis/MultiSigWallet/contracts/MultiSigWallet.sol.* Retrieved from Github: https://github.com/gnosis/MultiSigWallet/blob/master/contracts/MultiSigWallet.sol

Giuseppe Destefanis, A. B. (2018, March 29). *Smart Contracts Vulnerabilities: A Call for Blockchain Software Engineering?* Retrieved April 5, 2018, from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8327567

Grech, A., & Camilleri, A. F. (2017). *Blockchain in Education - Pedocs Open Access* . Retrieved 12 15, 2018, from https://www.pedocs.de/volltexte/2018/15013/pdf/Grech_Camilleri_2017_Blockchain_in_Education.pdf

Greif, S., Benitte, R., & Rambeau, M. (2018, 12). *Front-end Frameworks*. Retrieved from State of Js 2018: https://2018.stateofjs.com/front-end-frameworks/overview/

Hannan, J. (2018, 06 29). *Why You Probably Shouldn't Use a Single Private Key to Secure Your Assets*. (Medium) Retrieved 12 15, 2018, from https://medium.com/modular-network/why-you-probably-shouldnt-use-a-single-private-key-to-secure-your-assets-cc7df87b30a

Hwong, C. (2017, 10 11). *Chart of the Week: Which Devices Are Used Most Often?* (Verto analytics) Retrieved 12 16, 2018, from https://www.vertoanalytics.com/chart-of-the-week-which-devices-are-used-most-often/

id.ee. (2019, 03 18). *ID-card and Digi-ID, Article ID: 30500* . Retrieved April 2019, from https://www.id.ee/index.php?id=30500

International Transport Forum. (2018, May 16). *OECD iLibrary.* Retrieved 12 10, 2018, from https://www.oecd-ilibrary.org/transport/blockchain-and-beyond_bf31443f-en

Internet Engineering Task Force, (IETF). (2016, November). *PKCS #1: RSA Cryptography Specifications Version 2.2.* Retrieved from Internet Engineering Task Force: https://tools.ietf.org/html/rfc8017#section-8.2.2

Jalukse, K. (2019, 02 22). *Smart-ID documentation.* Retrieved from Github.com: https://github.com/SK-EID/smart-id-documentation#43-certificate-choice-session

King, R. (2018, August 21). *BitDegree*. Retrieved January 20, 2019, from https://www.bitdegree.org/tutorials/metamask/

Kubach, M. a. (2015). SSEDIC.2020 on mobile eid. In D. A. Hühnlein (Ed.), *Open Identity Summit 2015* (pp. 29-41). Bonn: Gesellschaft für Informatik e.V.

Ledger. (2018, 11 12). *What Can a Backup Device Do for You?* Retrieved 12 12, 2018, from https://www.ledger.fr/2018/11/12/what-can-a-backup-device-do-for-you/

Lõugas, H. (2018, 11 08). *digigeenius*. Retrieved 04 2019, from https://digi.geenius.ee/rubriik/uudis/suur-uudis-smart-id-saab-vordseks-omakaelise-allkirja-ja-id-kaardiga/

Logvinov, L. (2018, September 9). *Ethstonia Identity*. Retrieved January 30, 2019, from https://devpost.com/software/ethstonia-identity

Logvinov, L. (2019, 03 20). *LogvinovLeon/estid-sig: Verify Estonian e-id signatures on Ethereum*. Retrieved from Github: https://github.com/LogvinovLeon/estid-sig

Massanet, A. (2018, 12 22). *SolRsaVerify*. Retrieved from Github: https://github.com/adriamb/SolRsaVerify/tree/master/contracts

Mohanty, D. (2018). Advanced Programming in Oraclize and IPFS, and Best Practices. In *Ethereum for Architects and Developers* (pp. 151-179). Apress, Berkeley, CA.

Murthy, M. (2017). *Life Cycle of an Ethereum Transaction*. (Medium) Retrieved 12 12, 2018, from https://medium.com/blockchannel/life-cycle-of-an-ethereum-transaction-e5c66bae0f6e

Nõmmik, A. (2017, 05 19). *Smart-id-documentation.* Retrieved from Github: https://github.com/SK-EID/smart-id-documentation/blob/master/images/RP_API_sequences_REST.png

Politsei- ja Piirivalveamet. (2019, 04 10). *Identity Documents*. Retrieved 04 10, 2019, from https://www2.politsei.ee/en/teenused/riigiloivud/riigiloivu-maarad/isikut-toendavad-dokumendid/index.dot

Republic of Estonia Information System Authority. (2019, 01 09). *Electronic Identity eID*. Retrieved April 2019, from https://www.ria.ee/en/state-information-system/electronic-identity-eid.html

RF Wireless World. (2012). *traditional storage vs cloud storage | difference between traditional storage and cloud storage*. Retrieved 12 12, 2018, from http://www.rfwireless-world.com/Tutorials/traditional-storage-vs-cloud-storage.html

Saurel, S. (2018, 01). *Create your own Blockchain in 30 minutes*. (Medium) Retrieved 12 10, 2018, from https://medium.com/@ssaurel/create-your-own-blockchain-in-30-minutes-dbde3293b390

Sibold, G. (2018, 10 26). *digigeenius*. Retrieved 04 10, 2019, from https://digi.geenius.ee/rubriik/uudis/suur-ulevaade-smart-id-ja-mobiil-id-kasvavad-eestis-muhinal/

SK ID Solutions AS. (2018, 10 01). *Price List of Validity Confirmation Services*. Retrieved 04 10, 2019, from https://www.sk.ee/teenused/hinnakiri/kehtivuskinnituse-teenus/

SK ID Solutions AS. (2019). *SK services*. Retrieved March 2019, from https://sk.ee/en/services/

SK-EID. (2019, February). *Smart-ID decumentation*. Retrieved from Github: https://github.com/SK-EID/smart-id-documentation

Smart-ID. (2019, 04 11). Retrieved 04 2019, from https://www.smart-id.com

Unchained Capital. (2018, 03 08). *A simple & safe multisig Ethereum smart contract for hardware wallets*. (Medium | Unchained Capital) Retrieved 12 15, 2018, from https://blog.unchained-capital.com/a-simple-safe-multisig-ethereum-smart-contract-for-hardware-wallets-a107bd90bb52

Unchained Capital. (2018, 03 8). *A simple & safe multisig Ethereum smart contract for hardware wallets.* Retrieved from Unchained Capital: https://blog.unchained-capital.com/a-simple-safe-multisig-ethereum-smart-contract-for-hardware-wallets-a107bd90bb52

Unibright.io. (2017, 12 07). *Blockchain evolution: from 1.0 to 4.0*. (Medium.com) Retrieved 12 1, 2018, from https://medium.com/@UnibrightIO/blockchain-evolution-from-1-0-to-4-0-3fbdbccfc666

Valenta, M., & Sandner, P. (2017, June). *FSBC Working paper.* Retrieved from Frankfurt School Blockchain center: https://pdfs.semanticscholar.org/00c7/5699db7c5f2196ab0ae92be0430be4b291b4.pdf

Vujičić, D., Jagodić, D., & Ranđić, S. (2018). Blockchain technology, bitcoin, and Ethereum: A brief overview. *2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH).* East Sarajevo: IEEE.

Wilma Woo. (2018, 10 03). *$8 Billion Coinbase Now Has More Users Than 21 Million Bitcoins*. Retrieved from Bitcoinist.com: https://bitcoinist.com/8-billion-coinbase-now-has-more-users-than-21-million-bitcoins/

License

Non-exclusive license to reproduce thesis and make thesis public.
I, Aleksandr Tsõganov

1. herewith grant the University of Tartu a free permit (non-exclusive license) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,
   Integrating User Identity with Ethereum Smart Contract Wallet supervised by Orlenys López Pintado, Aivo Kalu and Kristjan Kuhi.
2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons license CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.
3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.
4. I certify that granting the non-exclusive license does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Aleksandr Tsõganov
14/08/2019