Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 221 (2023) 1074–1082

Procedia Computer Science

www.elsevier.com/locate/procedia

Information Technology and Quantitative Management (ITQM 2023)

# Digital Identity Using Blockchain Technology

Alexandru-Cristian Careja[a], Nicolae Tapus[a]

[a]University Politehnica of Bucharest, Romania

## Abstract

In recent years, the development of blockchain technology has brought a wide spectrum of potential application scenarios from finance to healthcare, IoT, management and many more. This is because the blockchain, a distributed digital ledger, acts as an environment of trust and transparency. The subject of digital identity is getting more and more attention as it represents more than just replacing physical documents (e.g. ID cards, driver's license) with digital data. It would enable the digitalization and automation of the process of identification as well as protect people against identity theft and fraud. This paper proposes and implements a model for digital identity that relies on blockchain technology and cryptography to ensure the privacy, validity and integrity of personal user data. These are obtained with the help of cryptographic signatures, which are the digital representation of physical signatures or stamps. A set of trusted authorities are in charge of issuing cryptographic signatures and storing them on the blockchain. Through issued cryptographic signatures, the authorities account for the digital identity of the users, similar to how public institutions issue identity documents for their citizens which prove their identity. Using digital identity in multi-participant decision support systems improves the decision-making process through trust, anonymity and interoperability.



Keywords: Blockchain; Identity; Privacy; Cryptography; Elliptic Curve; Signature; Hash; Public/Private Key

## 1. Introduction

Nowadays, interacting with most public services (e.g. healthcare, education, banking) requires a valid proof of legal identity. This proof often comes in the forms of ID cards, birth certificates or driver's licences. One of the downsides of this identity management model is that physical documents are subject to loss, theft and fraud. In the era of digitalization, if people could have a digital identity, it would enable the development of automated processes, reducing significantly the amount of bureaucracy that needs to be done. Another benefit from implementing a model for digital identity is represented by the increased security level that it provides, as digital identities can be protected through cryptography.

The blockchain technology can be likened to a decentralized database that runs on a peer-to-peer network. The same exact copy of the database is replicated across all the nodes of the network, that work in a consensus driven environment. Thanks to its nature, it can store data and guarantee that it

*Nicolae Tapus. Tel.: +4-074-452-3364.
E-mail address: nicolae.tapus@upb.ro.

1877-0509 
10.1016/j.procs.2023.08.090

will not be tampered with and it can prevent attackers from adding new data to the blockchain. It relies on computational power and cryptographic tools to keep its resources safe.

This paper aims to lay the foundations of a digital identity model that uses a decentralized, open-source database (e.g. blockchain) for transparency and security. It is mandatory that user's personal data privacy is maintained, therefore, instead of storing actual data, only a cryptographic proof of identity should be stored in the database, while the personal data should belong to the user only. This can be done with the use of hash functions and cryptographic signature schemas that will be described later in this paper.

The Ethereum blockchain's storage can be split into two parts: chain data which stores the list of blocks that form the blockchain, and state data that includes data like account balances and smart contract state variables. Most blockchains haven't been designed to store large amounts of data. However, there are a few blockchains that are focused around efficient storage in a transparent, distributed manner. An example of that is Storj [1], a crypto-powered cloud storage platform, with a claimed network capacity of over 100 petabytes (1 petabyte = 1024 gigabytes). What this blockchain aims to achieve is to allow users to pay other users on the network to store their files.

### 1.1. Digital Identity solution for blockchain

Several proof of concept papers [2, 3] describe the implications of digital identity in a decentralized environment. They emphasize the idea of "self-sovereign identity" (SSI), a model in which users own all the rights over their digital identity. In this model, users have control over what happens with their identity, as only they can share it with whoever they want to. In addition to these, the self-sovereign identity model supports the transparency of algorithms and systems, so that everyone can see how they work and have trust in the framework.

The SSI model is closely related to the Distributed Ledger Technology (DLT), and more specifically, blockchain technology. This is because the self-sovereign identity model needs to store cryptographic proofs that can be trusted by anyone and available at any time. The blockchain technology is a good foundation for SSI as it provides high level of security as well as scalability.

### 1.2. Hashes and hash functions

A hash function is a mathematical function which takes an input and returns an unique output (called hash) of fixed length. The input can be as small as a simple number or as big as a movie file. Hashes must be deterministic, so that the same input message always returns the same output hash. A small change in the input message should generate a completely different hash, therefore hashes must be uncorrelated. Finally, hashes must be unique; it should be hard to generate the same hash from two different input messages.

There are different types of hash functions, such as SHA1 (used by Git), SHA256 (used by Bitcoin and Ethereum blockchains), etc. These hash functions differ by the internal mathematics or algorithms that they use to convert the input to the output. Due to the use of different algorithms, the output of these functions is different for the same input. For example, the SHA256 is longer than the SHA1 hash. This is because the SHA1 creates a 40 digits hexadecimal hash (160 bits), while the SHA256 creates a 64 digits hexadecimal hash (256 bits). The Secure Hashing Algorithm 3 is used in blockchain applications for digital signatures, key derivation functions and others.

### 1.3. Digital signatures

A digital signature is a mathematical schema through which digital messages can be stamped by someone identified by a Public key and Private key pair. It is useful for verifying the identity of the originator, the authenticity of digital records as well as their integrity. The Digital Signature Algorithm (DSA), first described in 1992 [4] provides the basic signature generation and signature verification schemas. The DSA digital signature is a pair of large numbers represented in a computer as strings of binary digits. The Elliptic Curve Digital Signature Algorithm (ECDSA) implements the main functionalities of the DSA, and, in addition, it uses Elliptic Curve Cryptography for public-key and signature generation [5].

## 2. Solution Architecture

A digital identity model should be able to replicate most of the implications of the current identity model. One of the primary purposes of the current identity system is to authorize particular actions. When accessing a resource it is a must that the subject can be identified and prove that they have the right of using that resource. Identification is a way of establishing trust between individuals and is done by providing personal information (that is associated with the identity) and proof that the subject has the right to claim ownership of that identity. The proof is then analysed by the party that identifies the subject which decides whether to trust the individual or not.

### 2.1. The management of digital identity

The identity owner and the authorities are the two main actors in a model for digital identity, each of them having rights and responsibilities.

First, the identity owner should be in charge of the management of their personal data. This means that they has to protect the data themselves from being stolen or lost. The identity owner has the right to share their personal information whenever and to whomever they wish to, which in turn attracts a responsibility. Second, the trusted authorities are responsible for the authentication of user identities. They issue proof of authenticity and store it in a database, as well as manage those records.

### 2.2. Blockchain specifications

The blockchain that will be chosen needs to be a well established trustful distributed ledger that has proven in time that it is reliable and secure. From a technological point of view, it has to implement the following functionalities: a) be able to store large amounts of data; b) support smart contracts. Smart contracts help solve the issue of trust in the digital world.

### 2.3. Identification

So far, individuals have their own identities and authorities issue proofs of authenticity for user identities that are stored on the blockchain. These provide the foundation of identification. Identification is the process in which an individual tries to prove to an identifier entity, that their identity is legitimate. The process of identification should find a resolution only with the consent from the individual, as they own their personal data. Identification is usually requested when an individual tries to access a resource for which certain requirements are set, such as being an employee, or having the residency in a specific location.

### 2.4. Digital Signatures

There are a few security concerns over the subject of digital identity, one of them being, how can everyone be sure that the identity someone pretends to have is legit. One way to prove that a certain person's digital identity is legit is by having a trusted authority issue a cryptographic proof of authenticity for the user owned digital identity.

The Elliptic Curve Digital Signature Algorithm (ECDSA) uses the elliptic curve discrete logarithm problem (ECDLP), for which no sub-exponential time algorithm is known. Given a message and the private key of the signer, the ECDSA can compute a 64 bytes (sometimes 65 bytes) long digest (the signature). In some implementations of the ECDSA, it is required to implement the functionality of public key recovery from signature, which requires an extra byte of information, therefore making the signature 65 bytes long.

Elliptic curves define:

- A generator point $G$, used for scalar multiplication (for secp256k1 [6] elliptic curve, $n \approx 1.158e77$)
- Order $n$, the length of the private keys. (for secp256k1 elliptic curve, $G\{x \approx 5.506e76, y \approx 3.267e76\}$)

The ECDSA Private Key is generated as a random integer between 1 and $n-1$. The Public Key will then be generated as the elliptic curve multiplication between the Private Key and $G$. Given a key-pair($pubKey$, $privKey$), $G$, $n$ and a message $m$, the ECDSA Signature is generated following the next steps:

- Using the SHA3[1] calculate the hash of the message $m$: $hash = h(m)$;
- Calculate the value $k$ which is HMAC-derived from $hash + privKey$ as described in RFC6979[7];
- Calculate the abscissa $r$ of point $R = k \times G$;
- Calculate the signature proof: $s = k^{-1} \times (h + r \times privKey)(mod\ n)$, where $k^{-1}$ is the modular inverse of $k$;
- Return the signature $\{r, s\}$; $r, s \in [1, n-1]$.

The calculated $r, s$ integers encode the random point $R$ along with the proof $s$, showing that the signer knows the message $m$ and the Private Key $privKey$.

Based on these formalities, the decision to identify a person will be taken using cryptographic proof in the form of a digital signature generated using the ECDSA.

### 2.5. Modules architecture

The solution architecture is split into four modules, their relationship being illustrated in Figure 1.

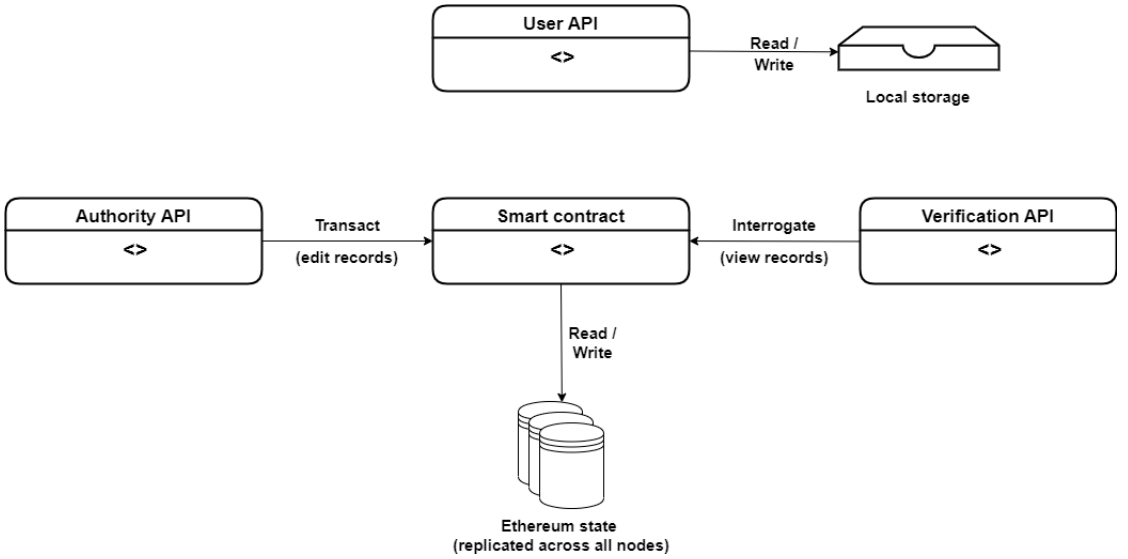


Fig. 1. Modules Architecture

The trusted authority will store the proofs of authenticity on the Ethereum blockchain using a Python API which sends transactions to the blockchain. These transactions will be processed by the smart contract which will verify whether the sender of the transaction is trusted to register a new signature and store them accordingly. Through this same API, the trusted authority can also edit and delete records, similar to how it can add new ones. The smart contract exposes methods for both authorities and verifying entities, each of them either public (the ones that don't alter the state of the smart contract) or restricted. Verifying entities will be able to interrogate the smart contract regarding the trustfulness of authorities as well as view authority issued signatures. On the user's side, a Python API will be implemented in order to help the user use the cryptographic tools needed for identification as well as help them manage their personal data. This module will use the local storage for storing the user's personal information.

[1] Secure Hashing Algorithm 3

### 2.6. Choosing the right blockchain

Choosing the right blockchain is crucial for an application like this. Designing and implementing an information system is a complex process that involves making many decisions related to system orientation, methodology and resources [8]. As mentioned in the specifications chapter, the chosen blockchain needs to support smart contracts and it needs to be able to store large amounts of data. Around 2014, blockchains started implementing smart contract functionalities, starting with the Ethereum blockchain. Other blockchains that support smart contracts are Solana, Neo, Cardano and a few more. However, none of these blockchains are really oriented towards storing tens of terabytes of data.

With that in mind, the blockchain that was chosen for implementing this model for digital identity is the Ethereum blockchain. It provides great documentation for its smart contracts as well as a few test networks (e.g. Ropsten) that can be used for application development. In addition, due to its popularity, developers have come to write libraries for various programming languages, making it compatible with most programming languages. Therefore, the smart contract is written in Solidity and due to the plethora of libraries for Ethereum transactions for Python, it will be the chosen language for programming the other modules. Ethereum however is not perfect, as the limitation this blockchain imposes is that it can not store large amounts of data.

### 2.7. Structural design of the digital identity

With the introduction of digital identity, people may ask "who is able to see my personal details?". In order to keep everyone's identity private, the subject's personal data will be stored on their personal devices such as computers, laptops, smartphones, tablets, smartwatches, or even flash drives. In this paper, the concept of digital identity will be defined by the sum of all attributes that an identity could have. The digital identity consists of:

- Actual personal data — One security concern that has been raised about the digital identity was who can see your personal details. In order to keep every identity private, the subject's personal data will be stored on their personal devices such as computers, laptops, smartphones, tablets, smartwatches, or even flash drives.
- Proof of ownership — Under the form of digital signatures (byte strings) stand as proof that the subject who signed it owns the data encrypted. This signature is critical in protecting data ownership because without it, a malicious subject could assume anyone else's identity, provided that they know the personal details.
- Proof of authenticity — Authority signatures (byte strings) that stand as proof of authenticity for user owned personal data. Tthey are meant to be available to anyone so they can validate a specific user's identity, while not divulging any personal data.

## 3. Implementation

### 3.1. Smart Contract

The smart contract module is written in Solidity, a programming language that exclusively targets the Ethereum Virtual Machine. The smart contract is a gateway for interacting with the blockchain state. Its purpose is to implement the functionality of holding signatures and provide a way to edit them. In the smart contract was defined a data structure for signatures which will hold the signature's bytes and a timestamp. The smart contract allows users to change the state of the blockchain. In this particular case, the state of the blockchain refers to the smart contract's state variables.

Adding new attributes is done through a smart contract function that stores a signature on the blockchain. This function is meant only for trusted issuers to run. If a non trusted party tries to call the function, it will lose the gas fees for the transaction without altering the state of the signatures mapping. The new attribute signature will be added to a signatures mapping if the issuer passes the trustfulness test and if the attribute is unique.

Updating and revoking attribute signatures also checks the request sender's trustfulness, but, require one additional step compared to registering new signatures. Before updating an attribute it has to be checked whether the issuer of the initial signature is the same as the one who tries to edit it. Therefore, comes the need of a function that recovers the issuer address from the signature, implemented in the smart contract as well.

In order for the signatures to be easily obtainable for verification purposes, the smart contract exposes a getter function which takes in as arguments the owner address and the attribute id and returns the corresponding signature.

After the compilation of the contract, a JSON file will be generated which contains the ABI (Application Binary Interface) that 'describes' to other applications how to interact with the contract's methods.

### 3.2. User API

In the implementation of the APIs (user, authority, verifier), were used the Web 3 Python library [9] for managing transactions and Keccak256 hashes due to its compatibility with the Ethereum blockchain. For calculating and verifying ECDSA signatures the solution uses the Ethereum Keys Python library [10].

The user API implements the generation of signatures over personal data. Signature generation is implemented in the 'sign' function illustrated in Figure 2. It receives one argument, the message (piece of information) that the user wants to sign. First, the Keccak256 hash of the message is calculated using the Web 3 library function 'solidityKeccak'. Then, the authority's private key is loaded and used for creating a PrivateKey instance, imported from the Ethereum Keys library. Finally, using the hash and the private key, generates the ECDSA signature.

```
def sign(msg):
    hash = web3.solidityKeccak(['string'], [msg]).hex()
    private_key = eth_keys.keys.PrivateKey(
        bytes.fromhex(os.environ['USER_' + user_no + '_PRIVATE_KEY'][2:]))
    signature = private_key.sign_msg(bytes.fromhex(hash[2:]))

    return signature.to_hex()
```

Fig. 2. User Signature Generation

### 3.3. Authority API

This API is designed for use for issuing authorities such as public institutions or even private companies. It implements the communication with the blockchain and provides methods for editing the blockchain state and for issuing ECDSA signatures. Similar to the User API, it uses the Web 3 and Ethereum Keys libraries for their cryptographic tools.

The first step in the implementation of this module was the transaction logic. Interaction with the smart contract is done through function calls, but because the functions calls from this module will modify the blockchain state, they will require to be part of a transaction, as they will cost an amount of coins to be processed.

The authority signature generation takes the attribute id, value and the user signature for that specific attribute. It calculates and returns the Keccak256 hash of the concatenation of the prefix, attribute id, attribute value and the user's signature. This hash will then be used as message that will be signed using the ECDSA. However, because the signature has to be compatible with the Solidity signature functions, one change has to be done to the signing algorithm from earlier. In solidity, the $v$ value of a signature has to be in the range $[27, 31]$. The Ethereum Keys library's ECDSA, calculates the $v$ value accordingly, but it returns $v - 27$. Therefore, after the signature generation, the $v$ value is restored by adding 27 to it, so that it makes the signature compatible with the smart contract.

Authority signature generation differs from the user signature generation process due to the fact that the authority signatures have to be compatible with Solidity's signature recovery mechanism. Therefore,

the authority signature will have to use the standard Solidity message prefix which can be broken into three elements:

1. the byte 0x19 - this byte would be an invalid first byte for a transaction, so its presence eliminates the risk of signing an unwanted transaction;
2. the string 'Ethereum Signed Message:\n';
3. the length in decimal numbers of the message that is signed (excluding the prefix length).

```
def getEthHash(id, attribute_value, user_signature):
    length = len(id) + len(attribute_value) + len(user_signature)
    eth_prefix = '\x19Ethereum Signed Message:\n' + str(length)
    eth_hash = web3.solidityKeccak(['string', 'string', 'string', 'string'],
        [eth_prefix, id, attribute_value, user_signature]).hex()

    return eth_hash
```

Fig. 3. Hash generation

The attribute registration function is used by authorities when issuing new signatures and storing them on the blockchain. This function receives the required data from input: the user's address, the attribute id, the attribute value and the user signature. It creates a function call to the according contract function with the specified arguments, and sends the transaction, by calling the 'transact' function. Updating and revoking signatures are implemented in a similar way to registering.

### 3.4. Verifier API

A Verifier API was implemented under the form of a class, so that any other Python app can import the class and use its methods. Because the calls that this module will send only refer to the smart contract's signature getter function, which is a 'view' function, it can not modify the state of the blockchain. For this reason, these function calls that read the state of the blockchain do not have to be included in a transaction. This class defines three new methods.

The Verifier API implements a method that does the verifications regarding the signature structure. Because the signatures are represented as hex strings until they are written on the ledger, the first check is to see whether the first two characters are '0x', then, it checks whether the signature has the right length and finally, it checks whether the signature is indeed a hexadecimal value, by trying to convert the value from base 16 to decimal.

The user signature validation method checks whether the given signature was issued by the given user address. It first checks the structure of the signature. If it passes this check, then, it calculates the hash of the signed message. Using the ECDSA pubic key recovery method, it determines from the signature and the signed message hash, the address of who signed the message. Finally, it checks the resulting address against the given address and takes the decision of whether to trust the user or not. If the two addresses are the same, then the identification process continues to the next and last step.

Based on the validity of the signature and the trustfulness of the user, the Verifier makes the decision of whether the identity is valid or not.

## 4. Digital Identity use in Multi-Participant DSS

A Decision Support System (DSS) [11] is an information system that supports business or organizational decision-making activities. Multi-Participant Decision Support Systems (DSS) should be anonymous so that ideas are accepted based on their values rather than the proposer's reputation or position [12]. Similar to how users share their personal data to only whomever they want, the proposed digital identity solution in adequate to be used in a DSS to first, identify the user within a decision-making task and secondly, submit votes anonymously.

Anonymous voting is achieved using the same signature schema that our digital identity solution uses. Voters sign their vote using their private key resulting in a hash which is then hashed with SHA3

and signed using an authority's private key. In this case, the authority which is signing the votes must also be counting them since it is the only one, apart from the voters who should be able to see the votes. Anonymity can be achieved if we assume that the authority signing the participant votes doesn't allow anyone to see the one-time signed vote.

To further automate the process of decision-making, using smart contracts in multi-participant DSS provide security and immutability that the blockchain technology offers. The smart contract shall take care of the vote counting, signing (as if it would be an authority) and ultimately taking the decision by consensus. Another benefit of using blockchain technology and digital identity in a DSS is that these provide the system interoperability and interconnectivity between applications. Participants can use the same identity across every other system, while the blockchain facilitates the connectivity between them.

## 5. Evaluation

The evaluation of the implemented solution is done with a scenario that tries to mimic a identification process. It assumes that a university has a few online resources and that students have personal 'accounts' with which they can identify and gain access to those resources. In the digital identity model that this paper describes, passwords are replaced by the ownership of the private key. Whoever owns the private key, can be assumed that they own the data encrypted with it. Therefore, these 'accounts' are nothing less than a given student id from the university (e.g. 'john.doe'), backed up by user (student) and university signatures.

In order to evaluate the solution, a server application was used, that represents a university's student platform, and which owns a couple of records (grades) for some student ids. It uses the Verifier API for user authentication and communicates with the User API over TCP. It listens for connections and when a connection is established it waits for a message that represents the user request. If the requested resource are the grades, then the server sends back a message in which it tells the user that they need to authenticate using their 'university_login'. Besides that, in the same message the server sends a 'secret', which it expects the user to sign in order to prove that they own the private key associated with the address they claim to have. This secret begins with the byte '0x19' so that it would be an invalid byte for a transaction. Then the server waits for the credentials and signed secret from the user. Upon receiving them, it checks the signed secret and if the signer matches the user address it proceeds to validate the user identity, using the Verifier API. If it passes this check, then it will send the grades to the user.

Applying the described evaluation method to the solution proved that the model is capable of providing a complete digital identity service. The security that this model provides is backed up by the SHA 3 and ECDSA schemes. The piece of personal data that has to be protected is first hashed with the Keccak 256 hashing function, resulting in a 256 bit digest, and then signed using the Elliptic Curve Digital Signature Algorithm. In order to recover the originally signed message from the data stored on the blockchain, it would be necessary for both the ECDSA and SHA3 to be broken, but for which there isn't any known vulnerability so far.

## 6. Conclusions

With the progress in the technology sector, more and more requests of a digital identity model have been raised. Digital identity is a hot topic at the moment, as researchers are looking for methods to implement safe, tamper-proof identity management models which keep the user data private. The blockchain technology provides a public, transparent, decentralized ledger that enables the foundation of a trusted environment thanks to its consensus mechanisms.

This paper proposed a digital identity model that uses the blockchain technology to store cryptographic proofs of authenticity for attributes that define the identity of a subject. These proofs are issued by a set of trusted authorities, similar to how public institutions issue identity documents for their citizens. In this model, the user's private key is the essential piece of information that allows them to use their identity. The private key, alongside the subject's personal information can be stored on any device

from computers and laptops to phones, smartwatches and even on physical cards and be used for authentication when needed. The digital identity's use in multi-participant DSS is to support anonymity, the voting and decision-making process and interoperability of the system.

The solution is a digital identity model which uses the blockchain's decentralized database to develop a transparent identity model, which keeps the user's personal information private. It allows users to identify themselves with the help of a signature schema that proves the authenticity and ownership of the data.

With the emergence of new blockchain technology, we could see this model projected onto another blockchain that has the capabilities of storing large amounts of data, while supporting smart contracts with high availability.

## References

[1] Storj whitepaper.
URL https://www.storj.io/storjv3.pdf
[2] M. Shuaib, N. H. Hassan, S. Usman, S. Alam, S. Bhatia, P. Agarwal, S. M. Idrees, Land registry framework based on self-sovereign identity (ssi) for environmental sustainability, Sustainability 14 (9). doi:10.3390/su14095400.
[3] A. J. Zwitter, O. J. Gstrein, E. Yap, Digital identity and the blockchain: Universal identity management and the concept of the "self-sovereign" individual, Frontiers in Blockchain vol. 3.
[4] The digital signature standard, Nist - Communications of the ACM, Vol.35, No.7.
[5] V. Kapoor, V. S. Abraham, R. Singh, Elliptic curve cryptography.
[6] Standards for efficient cryptography, SEC 2: Recommended Elliptic Curve Domain Parameters.
[7] Rfc6979, section 3.2.
URL https://datatracker.ietf.org/doc/html/rfc6979
[8] F. G. Filip, A decision-making perspective for designing and buildinginformation systems, IJCCC 7 (2).
[9] P. Merriam, J. Carver, Web 3 python library.
URL https://web3py.readthedocs.io/en/stable/
[10] Ethereum keys python library.
URL https://github.com/ethereum/eth-keys
[11] F. G. Filip, Creativity and decision support system, Researches in Computer and Informatics 1 (1).
[12] F. G. Filip, Collaborative decision-making: Concepts and supporting information and communication technology tools and system, IJCCC 17 (2).