A Blockchain Approach to Support Digital Contracts

Victor H. Breder, Laurival S. C. Neto, Thiago F. Medeiros, Ivan M. Padalko, Guilherme S. Oliveira, Vitor Venceslau Curtis and Juliana de Melo Bezerra
Department of Computer Science, ITA, Sao Jose dos Campos, Brazil

Keywords: Blockchain, Digital Contract, Digital Signature.

Abstract: Distributing computing that work with policies that flirt with democracy, in which parties can interact without an intermediary, has gained strength. In this way, an attractive idea is to support digital contracts by removing the third party and allowing the group to create and store contracts in a reliable and secure way, where contracts are immutable and easily retrievable. We propose a Blockchain approach to aid the management of digital contracts. The proposal considers contract encryption, digital signature, and protocols for chaining blocks with data related to digital contracts. We develop a prototype to ensure the viability of our proposal. We also present a case of use to demonstrate the prototype usage. We argue that proposal and implementation together create an appropriate environment for research and education purposes.

1 INTRODUCTION

Although the technology continues to advance, in recent years the computational capacity seems to been reaching some barriers, such as the difficulty to continuously scale down transistors and the problem of energy efficiency (Borkar and Chien, 2011). Such a scenario has found an alternative through distribution and parallelization. Distributed systems can be scaled up through the addition of more machines, which brings a greater tolerance to failures and allows resources' sharing throughout the nodes of the system. It's not trivial to design and verify the correctness of distributed algorithms. Fortunately, groundbreaking and innovative results are emerging, such as Google's Spanner (Corbett et al., 2013) and the revolutionary Bitcoin/Blockchain couple (Nakamoto, 2017).

The need to scale up has required new ways of thinking. Decentralized systems that work with policies that flirt with democracy, in which parties can interact without an intermediary, has gained strength. For instance, P2P sharing technology (Ding et al., 2004) and Blockchain technology (Miraz and Ali, 2018). Blockchain, in principle, works as a form of database. It assembles, in a specific order, blocks that contains data. Furthermore, consensus algorithms are be applied to create a distributed version of Blockchain in a way that security is guaranteed.

One interesting application of the distributed Blockchain idea is digital contracts, which are digital version of regular contracts (Cong and Zheng, 2017). In regular contracts, a group of interested people relies on a third party to validate a contract that verses about rules regarding something valuable to the group. After the approval of the involved ones about the contract terms, they sign the document, validate with the third party and store it. The idea of digital contracts in a Blockchain is to remove the third party and allow the group to create and store contracts in a reliable and secure way, where contracts are immutable and easily retrievable. Furthermore, it's even possible to create addendums to contracts stored in the Blockchain.

Regarding the use of Blockchain technology applied to contracts, literature explores the concept of smart contracts. A smart contract is in fact an auto enforceable code, running on top of a Blockchain, with rules that dictate how parties interact with each other. However, there is a lack of studies about scalability, performance and security of presented applications (Alharby and Moorsel, 2017). Kalamsyah et al. (2018) work with the idea of digital contracts, but they focus on the authentication process of contracts between two parties and in the importance of a witness process.

In this paper, we propose a Blockchain approach to aid the management of digital contracts. The proposal considers the contract encryption, the digital

Breder, V., Neto, L., Medeiros, T., Padalko, I., Oliveira, G., Curtis, V. and Bezerra, J.
A Blockchain Approach to Support Digital Contracts.
DOI: 10.5220/0007809804460453
In *Proceedings of the 14th International Conference on Software Technologies (ICSOFT 2019)*, pages 446-453
ISBN: 978-989-758-379-7

signature of parties, the validation of digital signatures, the storage of such information using block chaining. It is possible to deal with distinct contracts as well as with their possible addendums, always in a distributed manner. While keeping properties as integrity and authenticity of traditional contracts, our approach based on Blockchain eliminates the need of intermediaries and brings security to the whole process. Section 2 describes our proposal. Section 3 describes the evaluation of an implemented prototype. Section 4 presents conclusions and future work.

2 SUPPORTING DIGITAL CONTRACTS

In our proposal, users can, after a consensus, create a contract that is then inserted in the Blockchain. The network consists of several nodes that are aware of all the other nodes in the network, in other words, a complete network. Each node represents an instance capable of inserting blocks in the Blockchain. Each user group that wishes to create a contract does so through some node of the network. The proposal can be summarized into three distinct parts: digital contract creation and signing; Blockchain structure and the network protocols.

2.1 Digital Contract Creation and Signing

The digital contract is treated here as a pack of data. It's not necessarily needed to be stored in the Blockchain, in fact it is stored somewhere else. Only information that can be used to assert the contract is the same for all parties is needed, in other words, information that can assert the contract integrity. Furthermore, the contract is sealed only after all parties agree on its content with their respective signatures.

Nowadays, an effective way of checking data integrity is by using a hash function, a mathematical function that maps the data in such a way that even small changes to the original data causes a complete different result from the hash function. Usually, inverting the hash function isn't possible and collisions, when two different data contents are mapped into the same value by the hash function, are very rare, but can happen.

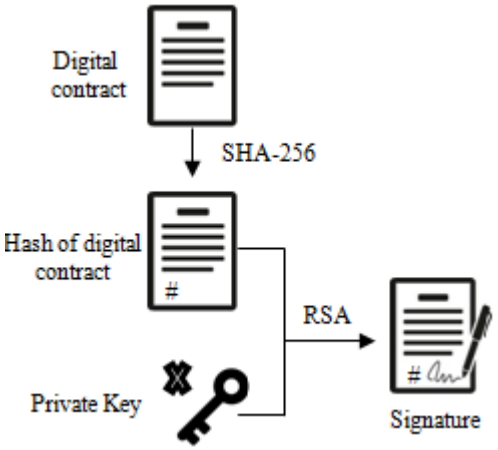

Figure 1: Signature generation scheme.

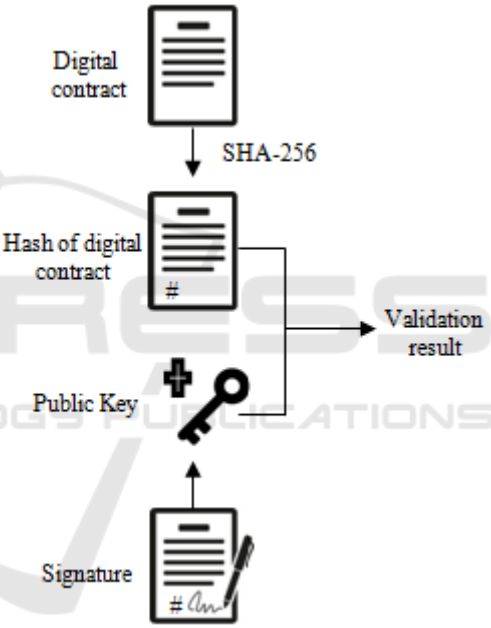

Figure 2: Signature verification scheme.

One of such functions is SHA-256, a hash function that, given input data of arbitrary size, produces a fixed 256 bit (or 32 bytes) word. Then, using SHA-256 it's possible for parties to check the contract integrity. Once every party agrees upon the contract, the hash of the document is used as a base for the signatures. This process uses the typical RSA authentication, where each party has a public key and a private key.

The process to generate a signature of a contract is depicted in Figure 1. Given the digital contract (for instance a file named 'contract.docx'), the SHA-256 function is used to generate a hash. The owner of a

private key then creates his signature of the hash by the RSA algorithm.

Conversely, given a signature and the correspondent public key, the signature can be verified. The scheme is shown in Figure 2. If the signature is authentic, operating with the public key over data that has been encrypted with the corresponding private key will decrypt and reveal the original data. The original data should be the hash of the contract that can be easily computed if the contract is available. So, if the result is the hash, the signature is valid.

2.2 Blockchain Structure

The Blockchain structure grows linearly as new pieces (blocks) are added. There is only one entry point for a new block. The addition of a block is made in such a way that consistency of the structure can be verified. The entire structure works as a register. The Blockchain structure provides a simple way to store data, but it's potential and interesting features are only apparent in the distributed form.

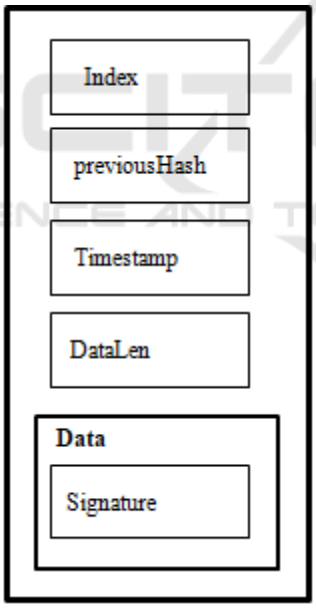

Figure 3: Block structure.

The block structure used in our approach is presented in Figure 3. The components are: an index that determines the position of the block in the chain; the hash of the previous block (computed with SHA-256 and using the block data); The timestamp; the data length (number of bytes of data); and the data itself. The data for our purposes consists only of a signature. In other words, each signature of a contract corresponds to a block in the Blockchain.

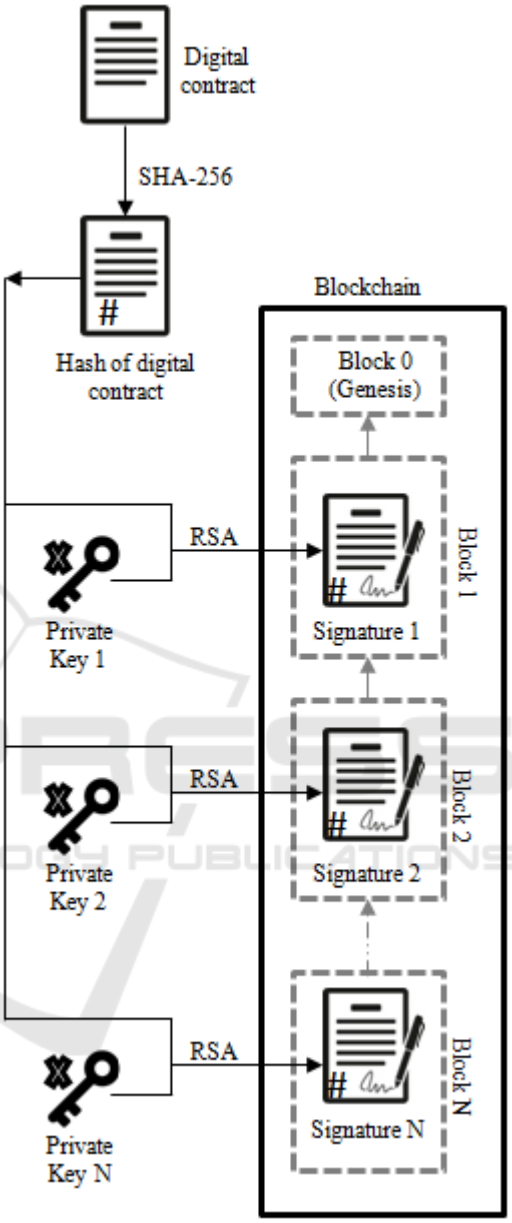

Figure 4: Blockchain formation scheme.

Regarding the chaining of blocks, the first block is the 'Genesis' block, one that does not has the hash of a previous block. After that, the chain can be built. A block can only enter in the Blockchain if it has the information about the hash of the last block added to it. This property is useful when considering the distributed case. Furthermore, the consistency of the chain can be easily checked by starting from the last

added block and verifying if the hashes match block to block until the 'genesis' block is reached.

A scheme of the Blockchain can be seen in Figure 4. Given a digital contract and its respective hash, each party that signs the hash, creates a block in the chain. It is important to remember that people can be distributed, so blocks can be originated in distinct nodes of the distributed system. The picture shows a chain with N blocks in order; however they could be in a different order given the distributed (and so out of sequence) characteristic.

Following the idea of signature verification shown in Figure 2, the verification process in Blockchain presented in Figure 5. So, every signature (inside a given block) can be verified using a public key. A match in the result indicates that that person (with the used public key) has signed the given contract (using its hash in fact).

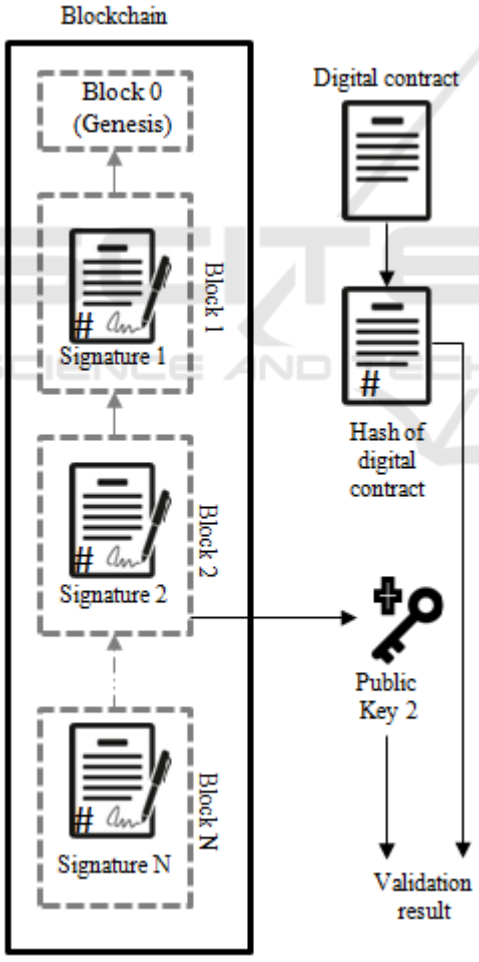

Figure 5: Signature verification in the Blockchain.

In case of contract addendums, the process is similar, as presented in Figure 6. Parties need to sign the hash of the addendum. New blocks are then generated and added in the current chain. In the same way, the Blockchain is up to register contracts and addendums of distinct groups.

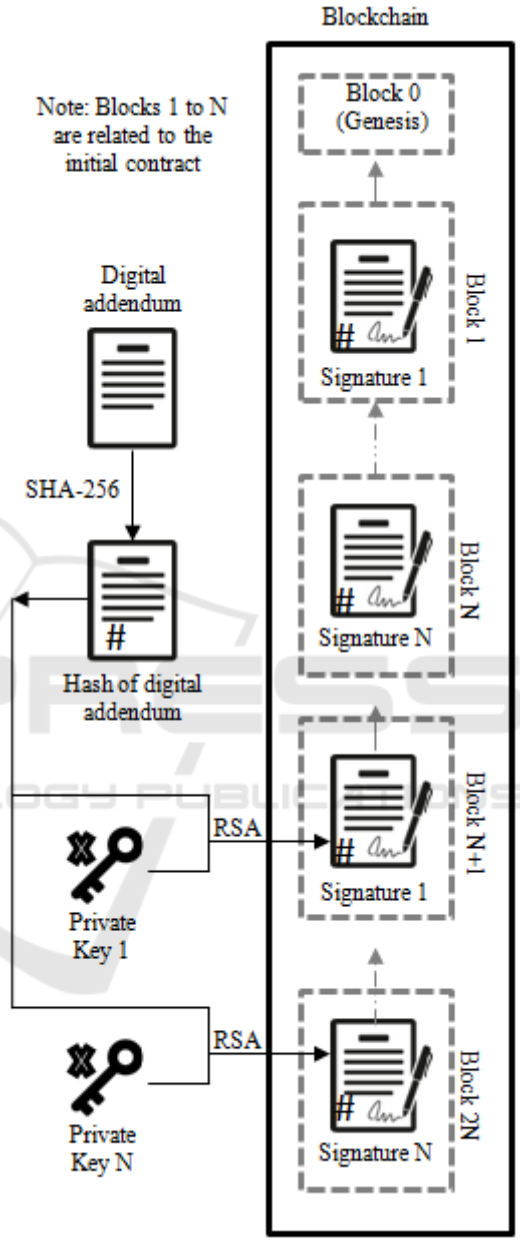

Figure 6: Contract addendums in the Blockchain.

2.3 Network Protocols

In our proposal, the system is a set of nodes in a network. Blocks can be generated by different nodes. Each node maintains a copy of the Blockchain. Here we discuss how nodes enter in the system, how blocks

are propagated among nodes, and how to manage eventual inconsistencies among chains in nodes.

Figure 7 shows a scheme for adding nodes in the network. When a new node B desires to enter the network, it must first establish a TCP connection with any of the present nodes already in the network (in the example, its node A). Node B sends a 'PEER-REQUEST' with its ID and Address to node A. Node A then acknowledge and store B's presence on the network (a local copy of existing nodes in the network). After processing the request, node A responds with 'PEER-ACCEPTED' and B can register A's presence (using its ID and ADDRESS) in the network as well.

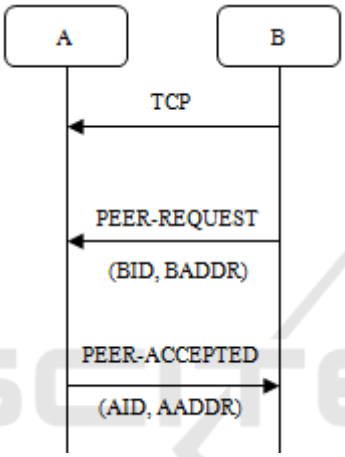

Figure 7: Protocol to add a new node in network.

The process of adding a new node continues until all nodes recognize the presence of the new one. The protocol is depicted in Figure 8. After the mutual acknowledgment in Figure 7, B sends a 'PEER-LIST' message to A, asking for the addresses of other nodes in the network. Node A, then, sends B sequentially 'PEER-ADD' messages. Each 'PEER-ADD' message has the address of a node of the network that B must connect to, by doing the same 'PEER-REQUEST' and 'PEER-ACCEPTED' iteration (already described in Figure 7). By the end of the process, the network remains a complete graph, in other words, every node is connected to every other node.

When a node (in the previous example, node B) enters the network, it also requests from the first node it connected (in the case, node A) a copy of the Blockchain. This process is presented in Figure 9. Node B sends 'REQUEST-BLOCKCHAIN' and A replies with many 'BLOCK-ADD' messages. Node A sends each block at a time on the same sequence it is stored locally. After that, B will have a copy of A's local Blockchain.

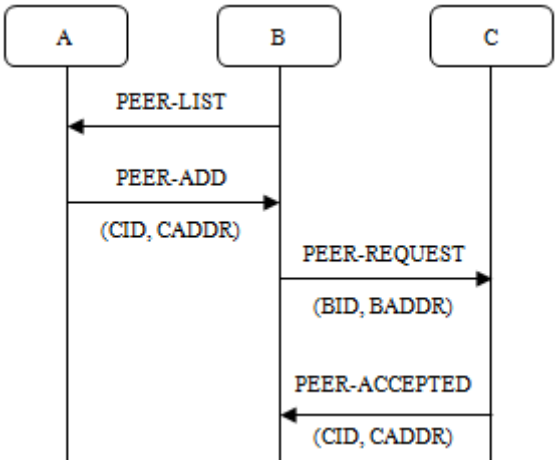

Figure 8: Protocol to inform network about a new node.

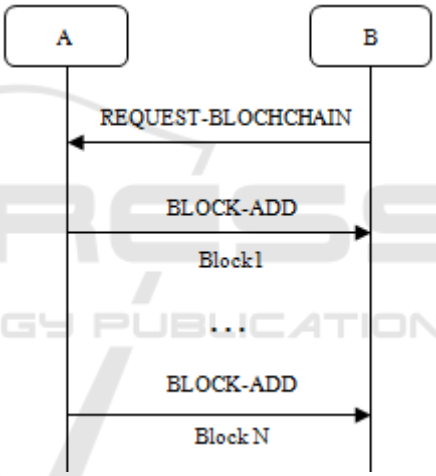

Figure 9: Protocol to copy the Blockchain between nodes.

Other protocol responsible for distributing the Blockchain is the broadcast of the addition of a new block, as shown in Figure 10. After node A insert a new block on its local Blockchain, it broadcasts the added block to the rest of the network. In a receiver node, if the new block fits as the next block (in other words, when it has the correct hash for the last added block on its local chain), then it is added at the end of the local chain. If in any case it doesn't fit the local Blockchain of a node, the new blocked is discarded in that particular node. Hence, it's possible to have inconsistencies on the consensus of the network about the Blockchain while adding multiple blocks at the same in different nodes and broadcasting it.

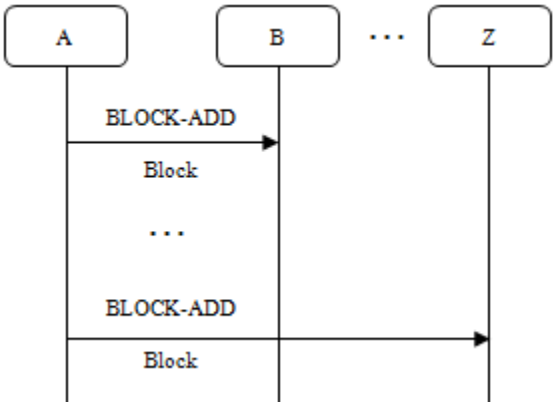

Figure 10: Protocol to broadcast a new block.

Lastly, to solve the pointed conflict of inconsistency of the consensus of the network about the Blockchain state, the nodes should follow a specific protocol. When a node receives a block with an index bigger than its own last inserted block, it requests from the sender the Blockchain. Since the Blockchain from the sender is longer, it gets prioritized and overwrites the local Blockchain of the requesting node. So, in the presence of a conflict, the longest Blockchain gets prioritized and the smaller chain is overwritten on the conflicting node.

This approach can lead to an exploit for malicious attacks, like creating a fake long chain and forcing the network to acknowledge it as the real Blockchain, since it's the longest. In fact, Bitcoin uses this approach, but to maintain the security of network and avoid this problem, it's also implemented the so called 'Proof-of-Work' or PoW (and there is a more recent idea about a 'Proof-of-Stake' or PoS), where the rate at which blocks are added to the chain is controlled and limited by computational power in such a way that trying to create a bigger chain in a small time interval would require a prohibitive amount of energy (Vashchuk and Shuwar, 2018). For this paper purposes, this potential problem is left unsolved for the sake of simplicity.

3 CASE OF USE

In a way to evaluate our proposal, we developed a prototype from scratch. It was implemented in the Go language and stored as an open-source project in a GitHub repository (link: https://github.com/impadalko/CES27Projeto). Here we describe a case of use, presenting existing commands and the demonstrating structures and protocols previously described.

The application starts as a single node with the 'Genesis' block created, as shown in Figure 11. The 'NodeId' is a random string and 'NodeAddr' is the address of node, so that other nodes can communicate and connect to it. The application can also be started with an address parameter. In this case, the node will try to connect to a node with the given address. An example is given in Figure 12, when the new attempts to connect to a preexisting node (in the case, with address 34369). The connection succeeds and it can be seen that the 'Genesis' block data is copied.

```
./CES27Projeto
NodeId:   WoQ57YZu
NodeAddr: [::]:34369

Index Hash     PrevHash Timestamp  Data
    0 e3b0c442 00000000 1543547516
```

Figure 11: First node instantiation.

```
./CES27Projeto :34369
NodeId:   W6GOogKr
NodeAddr: [::]:44713

Peer connected: WoQ57YZt

Block added:
Index Hash     PrevHash Timestamp  Data
    0 e3b0c442 00000000 1543547516
```

Figure 12: Initializing a new node.

```
peers

PeerId   PeerAddr
62TtJ0rH [::]:38819
W6GOogKr [::]:44713
```

Figure 13: 'peers' command.

```
conns

RemoteAddr      LocalAddr       PeerId
PeerAddr
127.0.0.1:53042 127.0.0.1:34369 W6GOogKr
[::]:44713
127.0.0.1:53044 127.0.0.1:34369 62TtJ0rH
[::]:38819
```

Figure 14: 'conns' command.

The 'peers' command (Figure 13) shows addresses of peers connected to the current node. The 'conns' command (Figure 14) shows, in details, the information of connections with current node. In these examples, another node was added (not shown

in previous figures). Generating public and private keys can be done with ‘genkey’ command followed by user nickname. Keys are stored as ‘.pem’ files in a local directory of the node. In this example, two pair of keys are generated, one to Alice (Figure 15) and other to Bob (similar to Figure 15).

```
genKey Alice

Generated private key Alice written to
Alice_priv.pem
Generated public key Alice written to
Alice_pub.pem
```

Figure 15: ‘genkey’ command to Alice.

```
privKey Alice

Using private key: Alice
```

Figure 16: Alice enables her private key.

```
hash contract.dat

The SHA256 hash of the file given is:
e3b0c44298fc1c149af...
```

Figure 17: Alice creates the contract hash.

```
sign e3b0c44298fc1c149af…

The document with hash e3b0c442 was
signed with key Alice and added to the
blockchain in block 1
```

Figure 18: Alice signs the contract hash.

The process of signing a contract hash is demonstrated below. In a given node, Alice uses ‘privkey’ command to enable its private key (Figure 16). Given a contract named ‘contract.dat’ stored locally, Alice creates its hash (Figure 17) and signs it (Figure 18). The signature is then stored as a block in the local chain (named as ‘block 1’) and later broadcasted to other nodes. In other node, Bob also signs the contract hash (Figure 19), generating ‘block 2’ in Blockchain as well.

```
sign e3b0c44298fc1c149af…

The document with hash e3b0c442 was
signed with key Bob and added to the
blockchain in block 2
```

Figure 19: Bob signs the contract hash.

```
blocks

Index Hash     PrevHash Timestamp  Data
    0 e3b0c442 00000000 1543547516
    1 2f715cac e3b0c442 1543547787 bc27076c
    2 1d9198e2 2f715cac 1543547832 d77b3c7b
```

Figure 20: Blockchain in a node.

```
pubKey Alice

Using public key: Alice
```

Figure 21: Alice enables her public key.

```
verify 1 e3b0c44298fc1c149af…

The signature is VALID
The document with hash e3b0c442 was
signed by Alice in the block 1
```

Figure 22: Alice verifies her signature.

```
add 1010

Index Hash     PrevHash Timestamp  Data
    0 e3b0c442 00000000 1543547516
    1 2f715cac e3b0c442 1543547787 bc27076c
    2 1d9198e2 2f715cac 1543547832 d77b3c7b
    3 430c0c4e 1d9198e2 1543548033 1010
```

Figure 23: Alice creates a block with data ‘1010’.

```
add 0101

Index Hash     PrevHash Timestamp  Data
    0 e3b0c442 00000000 1543547516
    1 2f715cac e3b0c442 1543547787 bc27076c
    2 1d9198e2 2f715cac 1543547832 d77b3c7b
    3 9dcf97a1 1d9198e2 1543548041 0101
```

Figure 24: Bob creates a block with data ‘0101’.

Blockchain can be seen in Figure 20. In fact, it is the chain stored in a given node, after adding Alice and Bob signatures of the hash of ‘contract.dat’. ‘Data’ column shows the partial signature data. Consider that Alice desires to verify its signature for the given contract. Firstly, she indicates the use of her public key (Figure 21). She then verify ‘block 1’ (created by her in Figure 18) against the contract hash (Figure 17). As expected, there is a match and the signature is authentic (Figure 22). In the same way, Alice and Bob can verify signature from each other in the Blockchain.

We demonstrate the process of tie breaking by using the longest chain. Consider that Alice creates a block with data ‘1010’ in a node. The block is stored locally and not broadcasted (Figure 23). Similarly, Bob creates a block with data ‘0101’ in a distinct node

(Figure 24). Both blocks are named as ‘block 3’ in the local chains. When block with data ‘0101’ is broadcasted to Alice’s node, it is ignored, since there is already a block with pointing to ‘block 2’ (with data ‘1010’).

```
add 0110

Index Hash     PrevHash Timestamp  Data
    0 e3b0c442 00000000 1543547516
    1 2f715cac e3b0c442 1543547787 bc27076c
    2 1d9198e2 2f715cac 1543547832 d77b3c7b
    3 430c0c4e 1d9198e2 1543548033 1010
    4 af340aa5 430c0c4e 1543548097 0110
```

Figure 25: Alice creates a block with data ‘0110’.

Later Alice creates other block, now with data ‘0110’. It results in a bigger chain shown in Figure 25. Finally, broadcasting the longest chain causes the overwritten of the other chain (with block ‘0101’). Hence, the implemented application behaves as the proposed idea of applying the distributed Blockchain algorithm to digital contracts.

4 CONCLUSIONS

We proposed an approach to support the management of digital contracts, motivated by the need to guarantee contract integrity and signature authenticity, and also to provide a trustable environment without involving intermediaries as with regular contracts. Our proposal is mainly based on Blockchain technology, but also includes contract hashing (using SHA-256 algorithm) and authentication based on public and private keys (using RSA algorithm).

The idea is that, after consensus about the contract rules, every part emits a signature of the hash of the digital contract. Each signature is inserted as a different block in the chain. This process happens inside a node of the network. When every party has signed the contract, the new blocks are broadcasted to the rest of the network. After some time, if the Blockchain is not overwritten by some other block addition, the agreement is done. If some signatures cannot enter the Blockchain, the local chain is updated and they are reinserted and broadcasted again, until all signatures are part of the Blockchain.

We developed a prototype to ensure the viability of our proposal. We also presented a case of use to demonstrate the prototype usage. The current examples can be easily generalized to any number of nodes, parties, contracts, and addendums. Extensions of our prototype are encouraged, in a way to facilitate the proposal application, for instance by adding facilities to communicate with processes across different machines, and by establishing directives to manage keys in a secure way.

Regarding the proposal itself, we intend to investigate enhancements in the protocols, including to deal with incomplete networks, and to add consensus algorithms for new blocks in the chain. We would like to study the proposal resilience to nodes’ failure and message losses. A web application to access Blockchain data is also of interest to improve the application usability. Both proposal and implementation are then suitable to further improvements, being an appropriate environment for research and education purposes.

REFERENCES

Alharby, M., Moorsel, A. V., 2017. Blockchain-based smart contracts: a systematic mapping study. *Computer Science & Information Technology (CS & IT)*, 7, 1-16.

Borkar, S., Chien, A., 2011. The future of microprocessors. *Communications of the ACM*, 54 (5), 67.

Cong, L., Zheng, H., 2017. Blockchain Disruption and Smart Contracts. *SSRN Electronic Journal*.

Corbett, J. C., et al., 2013. Spanner: Google’s globally distributed database. *ACM Transactions on Computer Systems (TOCS)*. 31 (3), No. 8.

Ding, H., Nutanong, S., Buyya, R., 2004. Peer-to-Peer Networks for Content Sharing. *Technical Report, GRIDS-TR-2003-7, Grid Computing and Distributed Systems Laboratory*. University of Melbourne, Australia.

Kalamsyah, S. A., Barmawi, A. M., Arzaki, M., 2018. Digital Contract Using Block Chaining and Elliptic Curve Based Digital Signature. *6th International Conference on Information and Communication Technology (ICoICT)*.

Miraz, M., Ali, M., 2018. Applications of Blockchain Technology beyond Cryptocurrency. *Emerging Technologies in Computing (AETiC)*, 2 (1).

Nakamoto, N., 2017. Centralised Bitcoin: A Secure and High Performance Electronic Cash System. *SSRN Electronic Journal*.

Vashchuk, O., Shuwar, R., 2018. Pros and cons of consensus algorithm proof of stake. Difference in the network safety in proof of work and proof of stake. *Electronics and Information Technologies*, 9.