Sexta Conferencia Nacional de Computación, Informática y Sistemas / CoNCISa 2018 / ISBN: 978-980-7683-04-3
Universidad de Los Andes, Mérida, Venezuela - 28 al 30 de noviembre de 2018

Proposals for a Smart Contracts Platform for Cryptocurrencies Based on the Mini-Blockchain Scheme

Carlos Sanguña, Sebastián Suarez, Antonio Russoniello, Miguel A. Astor
Universidad Central de Venezuela, Facultad de Ciencias, Escuela de Computación
Centro de Investigación en Comunicaciones y Redes CICORE
Los Chaguaramos, Caracas, Venezuela
{carlos.sanguna, sebastian.suarez, antonio.russoniello, miguel.astor}@ciens.ucv.ve

Wilmer Pereira
Universidad Católica Andrés Bello
Escuela de Ingeniería Informática
Montalbán, Caracas, Venezuela
wpereira@ucab.edu.ve

***Abstract*—The Blockchain data structure underlying Bitcoin and all related cryptocurrencies is designed to grow indefinitely in terms of the storage space required for it, a problem known as Blockchain bloat. A consequence of this problem is a tendency towards the centralization of Blockchain-based systems, as the Blockchain data structure becomes too unwieldy to handle for the majority of the users of the system. A solution to this problem is the Mini-Blockchain scheme proposed by J. D. Bruce and implemented by the Cryptonite Project, which limits the unbounded growth of the Blockchain by dividing it into three separate data structures, along with pruning policies for some of these data structures. However, the Mini-Blockchain scheme, as proposed, is unable to support smart contracts and related functionality that relies on the storage of persistent state beyond account balances. This paper presents two proposals for the incorporation of a smart contracts platform with persistent state into a Mini-Blockchain-based system, while keeping the stated storage scalability goals of the scheme.**

***Keywords*—Blockchain, cryptocurrencies, Blockchain bloat, Mini-Blockchain, smart contracts platform.**

I. Introduction

From their origins in the Bitcoin system specification by Satoshi Nakamoto [1], cryptocurrencies have presented themselves as an innovative decentralized alternative to the traditional financial system, while the Blockchain technology underlying these systems has gone on to affect a great number of industries as a disruptive technology for business processes that rely on centralized book-keeping [2]–[5].

However, Blockchain data structures suffer from scalability problems as a result of their design considerations [2]–[4]. In particular, being an append-only data structure means that Blockchains grow indefinitely over time, which creates problems for their local storage and sharing by users of a Blockchain based system. This problem also means that new nodes on a Blockchain based P2P (Peer-to-Peer) network will take longer times to bootstrap as the network ages, eventually making the use of full nodes an infeasible task for most users and contributing to the undesirable centralization of the Blockchain-based network [6]. This unbounded growth is known as the Blockchain bloat [7]. As an example, Figure 1 shows the increment in size of the Bitcoin Blockchain measured in Megabytes, from January 10, 2009 to January 19, 2018 [8].

A review of the bibliography reveals that very little research effort has been dedicated to find solutions to this problem [2]–[4]. In particular, only two solutions have been proposed so far: The Ultimate Blockchain Compression Project [9], which was abandoned in 2014; and the Mini-Blockchain scheme proposed by J. D. Bruce and the Cryptonite Project in [6] which is currently in active development.

The Mini-Blockchain scheme consists in the separation of the Blockchain functionality in three separate but related data structures, along with criteria that allows the system to regularly prune these data structures when the information they contain is no longer relevant to the functionality of the network. However, the Mini-Blockchain scheme as currently specified is unable to support the existence and execution of smart contracts due precisely to the pruning criteria used to limit the growth of the data structures central to the system. In our opinion, the lack of support for smart contracts greatly limits the viability of the Mini-Blockchain scheme as a scalable Blockchain system and cryptocurrency design, specially in light of robust smart contract solutions such as Ethereum [10], [11] and RSK [12].

The objective of this paper is to propose ways in which smart contract support can be added to the Mini-Blockchain scheme while maintaining the stated space scalability goals that brought the proposal of the scheme. In particular, we propose two different ways in which smart contracts can be added to the scheme, one by adding a new "contract tree" to the system linked with the Mini-Blockchain itself, and another that entails a modification of the current "account tree" data structure.

The rest of this paper is structured as follows. Section II describes the Mini-Blockchain scheme as a solution to the Blockchain bloat problem in the context of Bitcoin. Section III shows our two proposals to integrate Ethereum-like smart contracts to the Mini-Blockchain scheme, while maintaining its stated scalability goals. Finally, Section IV presents our conclusions regarding these proposals.

II. The Mini-Blockchain Scheme

In [6], with later revisions in [13], [14], J. D. Bruce presents the Mini-Blockchain scheme as a mechanism to reduce the rate

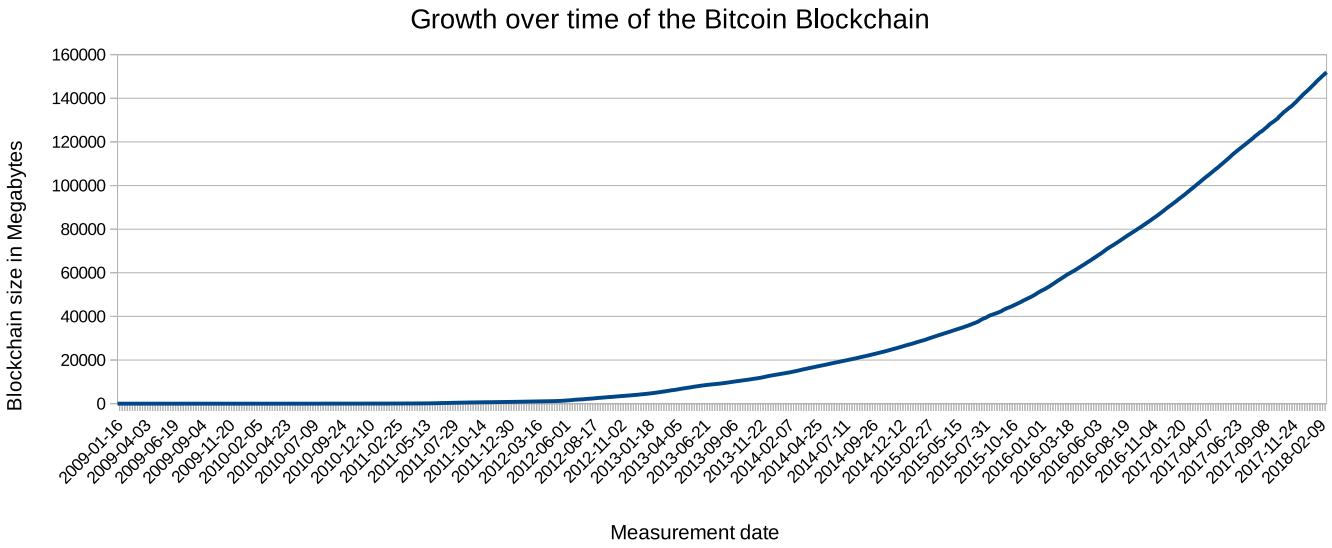

Figure 1: Growth of the Bitcoin Blockchain in Megabytes

of unbounded growth of storage space required for a Bitcoin-like Blockchain. This scheme is based on the observation by Bruce of how, in a network like Bitcoin, the Blockchain serves to fulfill three main objectives:

1) Coordinating the transaction processing by the nodes of the network.
2) Maintaining its own integrity via the Proof-of-Work (PoW) algorithm.
3) Storing and managing the account balances of all users of the network.

In particular, Bruce notes how it's not necessary to maintain the totality of the transaction history in the Blockchain forever when the transactions themselves are interpreted as simple operations on the account balances of the users. One way to visualize the functioning of the Mini-Blockchain scheme is then to think of the system not as a distributed ledger (as in Bitcoin), but rather as a distributed database of account balances. From this point of view, a transaction can be discarded once its operation is applied on the account balances of the participating users.

In short, the Mini-Blockchain scheme consists in separating the three functions of the Blockchain in three separate but related data structures. This three data structures are the account tree, the Mini-Blockchain itself and the proof chain [14].

A. Separation of Blockchain Functionality

The account tree consists of a binary search tree that combines a PATRICIA trie [15] with a Merkle tree [16], known simply as a PATRICIA-Merkle trie [11]. This data structure is used to store the accounts of the users. An account is composed of an address and a balance, among other parameters. The account tree covers the third functionality of the Blockchain.

The Mini-Blockchain is used to cover the first functionality of the Blockchain. This structure consists in a linked list of transaction blocks, where each block contains a header that includes the Merkle root of the account tree after applying the transactions included in the block, along with the hash of the previous block. When a new block is solved and added to the Mini-Blockchain, the nodes of the network apply the transactions included in the block to their own view of the account tree and verify its validity by comparing its Merkle root with the one included in the block. The general working of the Mini-Blockchain can be seen in Figure 2.

However, in order to guarantee the integrity of the network (the second function of the Blockchain) it's necessary to maintain the entire history of the application of the PoW algorithm, so that the nodes of the network are able to identify the version of the Mini-Blockchain that has been accepted by the users of the network. This is achieved with the proof chain, which consists in a linked list of the block headers from the mini-blockchain, as can be seen in Figure 2, adapted from [17]. The Mini-Blockchain itself contains the headers from the proof chain and a list of transactions processed for each corresponding block.

B. Pruning of the Mini-Blockchain and the Account Tree

Contrasting with the Blockchain scheme used by Bitcoin and other cryptocurrencies, the Mini-Blockchain is of finite length. The nodes of the network only store the last N blocks, where N is known as the Mini-Blockchain cycle time. This defines a window or view of the chain, and when a node no longer fits inside the window it can be discarded by the nodes[1]. This discarding mechanism has as a consequence that Mini-Blockchain networks have inbuilt micro-transaction[2] support and can store arbitrary custom messages in the transactions without fear of bloating the chain [14].

[1]The Mini-Blockchain's cycle time is a parameter of the network.

[2]A micro-transaction is a transaction that represents the transfer of a very low amount between accounts, possibly even less than what is paid as a fee for the processing of the transaction.

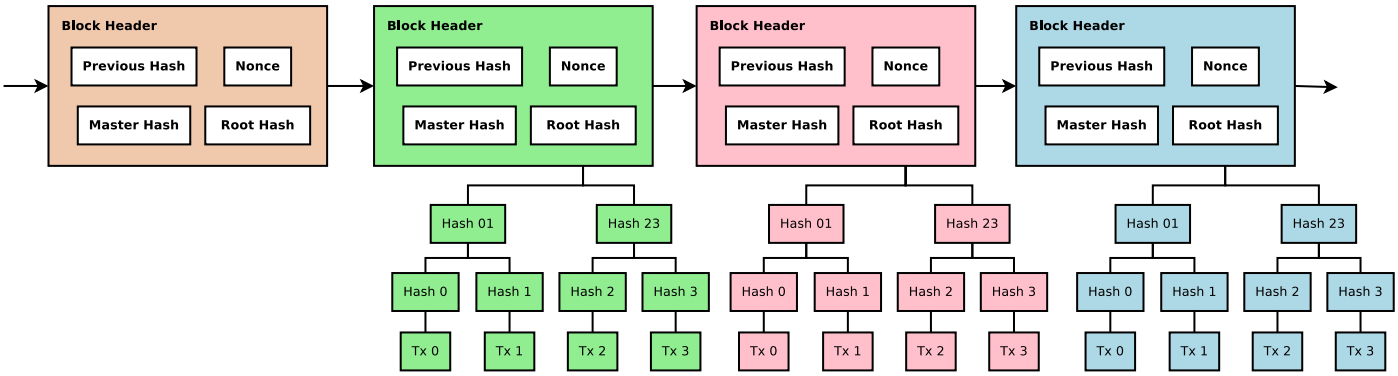

Figure 2: The Proof Chain and Corresponding Views of the Account Tree

The greatest use of storage space in the Mini-Blockchain scheme lies in the account tree, which can grow indefinitely and fill up with low valued or empty accounts. To mitigate this problem, Bruce proposes a pruning mechanism for the account tree [14], according to which the accounts that posses a balance of zero can be removed from the tree. Additionally, an "account maintenance fee" is charged to the origin account of each transaction. The intent of this fee is to increase the probability of pruning very low-valued accounts from the tree[3].

In the same way, the proof chain can also grow indefinitely. However, since this chain only stores the block headers, its storage requirements are considerably lower than what is required for a full Blockchain [14].

III. Adding Smart Contract Support to the Mini-Blockchain Scheme

This section introduces the concept of a "smart contracts" platform in a Blockchain-based network, followed by our proposals to incorporate such a platform to a Blockchain system based on the Mini-Blockchain scheme.

Extending or modifying the design criteria set by Nakamoto [1], it's possible to conceive the use of the Blockchain technology as a generic distributed consensus system for applications beyond monetary systems [10]. When a transaction is added to a block, the miner nodes execute programs embedded in the Blockchain as identified by the transactions. These programs can examine conditions related to the state of the accounts participating in the transaction (and the program's own account) and perform transfers of value and state updates, accordingly [2], [10].

In Bitcoin's case, these programs are designed in a Forth-like [18] language, called Script[4], known for being intentionally limited in its capabilities[5]. By removing these limitations[6], it becomes possible to design more sophisticated programs that allow to embed arbitrary business logic in the transactions of a Blockchain-based system [10]. These more sophisticated programs are known today as "smart contracts", though the term was originally introduced in 1996 by Szabo in [19], [20], long before the creation of Bitcoin. The first cryptocurrency to implement a smart contract mechanism like this is the Ethereum system [10], [11], which is possibly the most emblematic example of a smart contract platform today.

A. Outline of a Smart Contract Platform with Persistent State

In this section, we give a brief outline of a smart contracts platform with persistent, Blockchain-verified state in the style of the Ethereum system. In these kinds of system, two types of users can be identified: human users and software-based agents. Both kinds of users are associated to an account on the system and they can store, transfer value, and receive value from other users, of the same type or of the opposite type of user. The main difference between these users is that the software-based agents are associated to a smart contract that govern their interactions with human users or other software agents. Another chief difference is that software-based users are associated to a memory image that persists during the existence of the software agent, from its instantiation until its deactivation. The union of all the memory images from all software-based users and the account balances from all users of the system, software-based or otherwise, is known as the global state of the system [11].

The memory image of a software-based user is an array of bytes that can be used as a storage area for variable and data mappings managed by the agent's smart contract. This memory is stored externally from the Blockchain in a PATRICIA-Merkle trie. This data structure is linked to the Blockchain by including its Merkle root in the blocks. Alongside the memory image, this tree also stores a pointer to the account that belongs to the software agent in the data structure used to hold the account balance[7]. On the other hand, the account data structure for a software agent must store either the code of its corresponding smart contract or a pointer to another data structure where it is located. In Ethereum's case, the compiled bytecode of all the smart contracts in the system is stored in a hash table external to the account tree.

An Ethereum account includes then the hash value of its associated bytecode [11]. An account associated to a human

[3]The account maintenance fee is calculated dynamically during the run time of the network.

[4]A description of the Script language is available in https://en.bitcoin.it/wiki/Script.

[5]For example, Script doesn't include any form of jump instruction as a security measure.

[6]That is, using a Turing-complete language to write the Blockchain programs.

[7]Usually another PATRICIA-Merkle trie.

user contains the hash of the empty string as an indicator that it is not associated to any bytecode. Once a smart contract has been instantiated, it is added to the bytecode hash table and its hash is stored in its respective account. However, in Ethereum the code hash is never allowed to be modified after it has been set in an account, which means that its associated bytecode cannot change once it has been deployed to the network [11].

When two users communicate on a system such as this, they exchange a message[8] indicating the operation they wish to perform. This message can take one of two forms in the Ethereum system: it can represent a smart contract instantiation or it can be an exchange of value between two users. Both kinds of messages can trigger the execution of a smart contract's code if the recipient of the message is a software base agent, in the case of the second kind of message or always in the case of the constract instantiation message[9]. This execution is handled by a virtual machine that is instantiated by the system's client and mining software [21]. The definition and architecture of this virtual machine is beyond the scope of this paper.

With this outline we can determine the minimum components necessary for a smart contract platform with persistent state. More details about the specification of the Ethereum smart contract platform and associated virtual machine (EVM) can be found in the work of Gavin Wood [11] and Micah Dameron [21].

B. Proposals of Smart Contract Platforms for the Mini-Blockchain

As discussed in Section III-A, a smart contract platform with persistent state requires at least the following components:

1) A distinction between human users and software-based users.
2) A way to instantiate software-based users and deploy their respective program code to the network.
3) A way to store the smart contract program code associated to a software-based user.
4) A way to store the memory images of software-based users.
5) A mechanism to link a software-based user's account to its program code and memory image.
6) A way to execute the smart contracts when a transaction requires it.
7) A way to verify these components through the blockchain.
8) A way to disable or destroy[10] the program code of software-based users.

Additionally, a programming language and a compiler for it are required in order to produce the code of smart contracts. The details and specification of these tools are beyond the scope of this paper, however.

[8]In accordance with it's view of the system as a general purpose decentralized state machine, message is the Ethereum terminology for what other cryptocurrencies call a transaction [11].

[9]A contract instantiation implies the execution of the contract's constructor method, at least in the object-oriented programming model used by Ethereum.

[10]In the terminology of Ethereum, destroying a contract means disabling it completely.

Based on these requirements we identify two possible ways in which a smart contracts platform can be added to the Mini-Blockchain scheme. The first of these proposals consists of adding a new "contract tree" data structure, quite similar to the account tree, and adding the Merkle Root of the contract tree to the block header structure to link the contract tree with the Mini-Blockchain and the proof chain. On the other hand, the second proposal consists of adding the same functionality of the proposed contract tree to the account tree, in order to avoid a modification of the other data structures of the system.

1) Proposal 1: The Contract Tree: This proposal consists of an architecture quite similar to the specification of the Ethereum platform. It's core is a PATRICIA-Merkle tree we call the "contract tree" that would be used to store the memory images of all the smart contracts deployed to the network. Each memory image would be linked to a corresponding account on the account tree. The bytecode associated to each contract could then be stored either with it's memory image in the contract tree or, just like in Ethereum, in a separate data structure that would be also linked to the account tree.

In order to verify and secure the contract tree, it's Merkle root would be stored in the block headers of the Mini-Blockchain and the proof chain, just like the account tree.

2) Proposal 2: Modification of the Account Tree: Our second proposal consists of merging all these data structures together into the account tree. In this case the account tree would be extended to optionally include fields for program code and persistent memory in addition to the account balance and associated meta-data for the case of smart contract accounts. This has the advantage of needing modifications only for the account tree. No additional data structures are necessary, and the Mini-Blockchain and proof chain do not need to be modified.

The main disadvantage of this proposal is that in this case the account tree would bloat considerably with information that is beyond the scope of it's main responsibility, which is storing account balances.

3) Extensions to the Mini-Blockchain Transaction List: Regardless of the integration proposal, there is a need to extend the list of transactions proposed by the Mini-Blockchain scheme in order to completely support the requirements of a smart contract platform as specified. In this regard, we propose the addition of two new transactions to the three transactions already defined in the Mini-Blockchain specification:

- Contract instantiation.
- Contract execution.

The contract instantiation transaction would be used to perform all the steps necessary to deploy a new software-based user on the network, while the contract execution transaction would work to request the execution of a specific method of a contract and update the Blockchain state.

This would bring the amount of transactions required from a Mini-Blockchain implementation to five, including the value transfer, withdrawal limit modification and claim of block reward already defined in the Mini-Blockchain specification [14].

Another possibility could be extending the value transfer transaction to include information to distinguish when a transaction requires the execution of a smart contract. This would require just the addition of one new transaction to the scheme.

4) Execution of Smart Contract Code: The proposals outlined in Sections III-B1 and III-B2, along with the additions to the transaction lists described in Section III-B3 fulfill the requirements 1, 2, 3, 4, 5 and 7 of the requirement's list shown at the beginning of Section III-B. In order to fulfill requirement 6, it is necessary to incorporate a mechanism for contract code execution, that is, a virtual machine (VM) that will execute the compiled contract code. An example of a virtual machine for smart contract execution that fits the requirements stated in Sections III-A and III-B is, of course, the Ethereum Virtual Machine (EVM), designed by the Ethereum Project and described in [11]. Our proposals, however, can be implemented independently of the specific VM used to execute contract code.

5) Destruction of Smart Contract Code: The last remaining requirement of a smart contract platform is requirement 8: a way to disable or destroy the smart contract code already deployed on the network. In the case of proposal 1, disabling a smart contract would entail it's removal from the contract tree while transferring the remaining balance to the account that owns the contract. This transfer would empty the account of the contract and it would be pruned as specified by the Mini-Blockchain scheme. This behavior is consistent with the smart contract semantics of the Ethereum platform.

Regarding proposal 2, another disadvantage to it lies precisely in considering what happens when an account associated to a smart contract is either disabled or has a balance of zero. Under the specification of the account tree this account would need to be pruned, along with it's program code and memory image, which is not necessarily correct, since having a balance of zero doesn't imply that a contract should be disabled completely, it should still be available to receive funds. The inverse, disabling a contract with balance greater than zero, could be fulfilled in the same way as in proposal 1.

IV. Conclusions

This paper presents an overview of the Blockchain bloat problem and describes Bruce's Mini-Blockchain solution, currently in active research and development [14]. Blockchain bloat is a condition that arises due to the immutable nature of the Blockchain data structure. As specified, the Mini-Blockchain scheme is unable to support features like smart contracts that require the storage of persistent state beyond account balances.

Based on this, this paper proposes two ways in which support for a smart contract platform could be added to the Mini-Blockchain scheme, based on the design of the Ethereum smart contract platform. The first proposal consists on the addition of at least one new data structure, dubbed the "contract tree", that would be used to store the persistent memory image of the smart contracts and possibly the program code of the contracts itself. This contract tree would be linked to the account tree and would be verified by storing it's root hash in the block headers of the Mini-Blockchain. The second proposal consists of storing the memory image and program code of the smart contracts on the account tree itself, which requires less modifications to a Mini-Blockchain-based system, but brings about a series of delicate considerations and edge cases that the previous proposal lacks.

As described in Section III-B, our proposals fulfill all the requirements of a smart contracts platform. However, our second proposal outlined in Section III-B2 presents some semantic difficulties that make it nonviable in light of our other proposal described in Section III-B1. Based on this, the authors have started a proof-of-concept implementation of proposal number one to be integrated to the Cryptonite codebase, using the C++ implementations of the Solidity compiler and the EVM from the Ethereum project.

References

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," *white paper*, 2008.

[2] F. Tschorsch and B. Scheuermann, "Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 2084–2123, 2016.

[3] I.-C. Lin and T.-C. Liao, "A Survey of Blockchain Security Issues and Challenges." *IJ Network Security*, vol. 19, no. 5, pp. 653–659, 2017.

[4] J. Yli-Huumo, D. Ko, S. Choi, S. Park, and K. Smolander, "Where is Current Research on Blockchain Technology?—A Systematic Review," *PloS one*, vol. 11, no. 10, p. e0163477, 2016.

[5] J. A. Bergstra and K. de Leeuw, "Questions Related to Bitcoin and Other Informational Money," *arXiv preprint arXiv:1305.5956*, 2013.

[6] J. D. Bruce, "Purely p2p crypto-currency with finite mini-blockchain," https://bitcointalk.org/index.php?topic=195275.0, 2013.

[7] A. Wagner, "Ensuring Network Scalibility: How to Fight Blockchain Bloat," *Bitcoin Magazine*, vol. 6, 2014.

[8] Bitcoin.com, "Blockchain Size | Bitcoin.com Charts," https://charts.bitcoin.com/chart/blockchain-size#e8, 2018.

[9] A. Reiner, "Ultimate Blockchain Compression," https://bitcointalk.org/index.php?topic=88208, 2012.

[10] V. Buterin, "A Next-Generation Smart Contract and Decentralized Application Platform," *white paper*, 2014.

[11] G. Wood, "Ethereum: A Secure Decentralised Generalised Transaction Ledger," *Ethereum Project Yellow Paper*, 2014.

[12] S. D. Lerner, "RSK White Paper Overview," https://www.rsk.co/, 2015.

[13] J. D. Bruce, "The Mini-Blockchain Scheme, Rev. 2," https://cryptonite.info/files/mbc-scheme-rev2.pdf, 2014.

[14] J. D. Bruce, "The Mini-Blockchain Scheme, Rev. 3," https://cryptonite.info/files/mbc-scheme-rev3.pdf, 2017.

[15] D. R. Morrison, "PATRICIA—Practical Algorithm to Retrieve Information Coded in Alphanumeric," *Journal of the ACM (JACM)*, vol. 15, no. 4, pp. 514–534, 1968.

[16] R. C. Merkle, "A Digital Signature Based on a Conventional Encryption Function," in *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 1987, pp. 369–378.

[17] Cryptonite.info, "Mini-Blockchain," http://cryptonite.info/wiki/index.php?title=Mini-blockchain, 2014.

[18] P. L. Seijas, S. J. Thompson, and D. McAdams, "Scripting Smart Contracts for Distributed Ledger Technology." *IACR Cryptology ePrint Archive*, vol. 2016, p. 1156, 2016.

[19] N. Szabo, "Smart Contracts: Building Blocks for Digital Markets," http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html, 1996.

[20] N. Szabo, "Formalizing and Securing Relationships on Public Networks," *First Monday*, vol. 2, no. 9, 1997.

[21] M. Dameron, "Beigepaper: An Ethereum Technical Specification," *Ethereum Project Beige Paper*, 2018.